Digital Lab Assistant

Done by

Harshitha. N and R. Sai Sreya

Class 12F1 – February 2023

BONAFIDE CERTIFICATE

Certified to be the bonafide project work done by

Master / Miss _____

of Class _____ in PADMA SESHADRI BALA BHAVAN SR. SEC. SCHOOL, CHENNAI.

During the year _____

Date _____ P.G.T. in _____

Chennai

Submitted for All – India Senior Secondary Practical held in

_____ at

_____ Chennai.

Date _____ Examiner

Seal

Acknowledgment

At the outset, I wish to submit that besides, the sincere efforts of myself. I owe my gratitude to everyone who has been instrumental in the successful completion of our Project I take this opportunity, first and foremost, in acknowledging, with a deep sense of reverence. my profound sense of gratitude to our Computer Science Teacher Mr. Praveen Venkatachari- who has helped us formulate ideas and translate them into tangible results and further endeared us to improve upon the contents of the project. Without his able guidance. invaluable insights, encouragement, and support, this project would not have achieved its objectives.

Secondly, I wish to specially thank my project- partner and good friend- R. Sai Sreya, for the great deal of effort and time that she invested. I deeply admire and appreciate her stimulating ideas, active guidance, help, and cooperation which has mutually benefitted both of us.

Last but not the least, I'm greatly obliged to thank my friends and parents for providing us with the necessary moral support which has been a motivational factor in the completion of the project.

Index

Introduction

The Digital Lab Assistant- is a graphical user interface that provides users with the ease and convenience of learning science experiments over a screen. It has been developed to supplement the traditional physical lab manuals and bridge the constraints of the availability of study material.

It aids the user in performing the experiments by providing the procedure that needs to be executed sequentially and additionally, also lists the possible sources of error and the precautions that the user needs to ensure.

The program also performs all the calculations required to arrive at the desired result. Furthermore, it allows user to switch between the several tabs available (from any given frame), thereby establishing an efficient navigation across the interface.

Scientific data comes in many forms, from many different sources. Digital Lab Assistants are specifically designed to integrate scientific data and simplify workflows for scientists, lab managers and students. They can access and capture data hands-free at the point of experimentation, allowing real-time data-driven decisions. The calculations done by the program saves time and reduces error rates.

Objective and Scope

The objective of this project is to develop an interface that guides the user in performing science experiments that infer many crucial laws in Physics and Chemistry. It lists the procedure that the user is required to carry out and performs all the calculations involved to arrive at the result.

The scope of the project is restricted to the syllabus of classes 11 and 12 as prescribed by the CBSE.

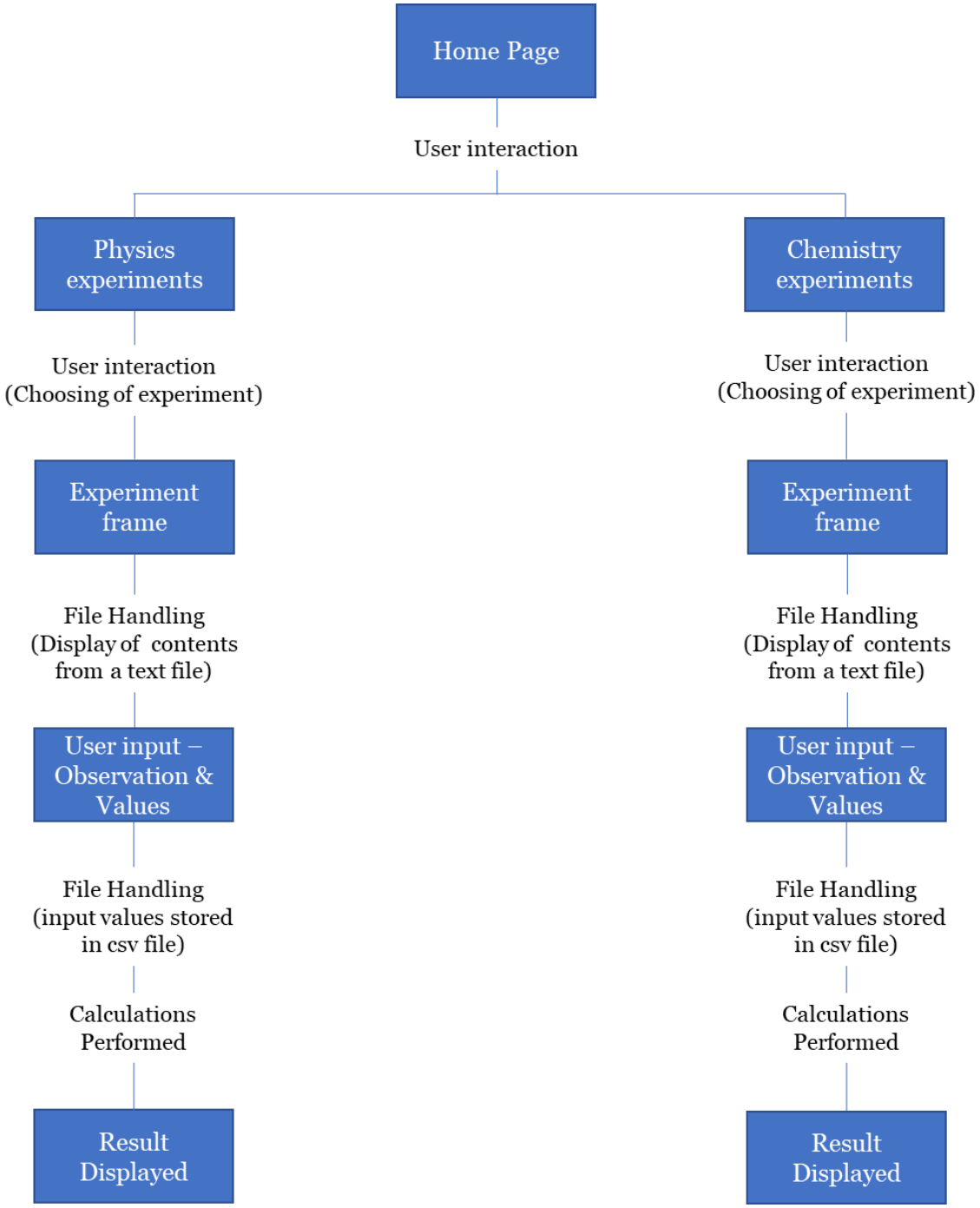
The experiments covered are:

Concave Mirror, Convex Lens, Ohm's law, Meter Bridge, Meter Bridge (Series and Parallel), Galvanometer, Convex Mirror, and Diode- in Physics and

Estimation of Potassium Permanganate using FAS, Estimation of Potassium Permanganate using Oxalic Acid, Rate of Reaction- Effect of Temperature, and Rate of Reaction- Effect of Concentration- in Chemistry.

System Design

- The 3 main frames of the interface is- Main home page, Chemistry home page and Physics home page.
- The user can switch to the physics and chemistry homepage from any given frame (experiment pages) using the respective buttons.
- The exit button allows the user to close the interface.
- The system design is as follows:



List of Datasets

Name	Type	Purpose
VolmFAS.csv	CSV file	Used to store values of molarity of FAS and concordant volume.
VolmOxalic.csv	CSV file	Used to store values of molarity of oxalic acid and concordant volume.
rate_temp.csv	CSV file	Used to store values of temperature and time.
rate_conc.csv	CSV file	Used to store values of volume of sodium thiosulphate and water.
concaveMirror.csv	CSV file	Used to store readings of position of object, mirror and screen as observed by the user.
convexLens.csv	CSV file	Used to store readings of position of object, lens and screen as observed by the user.
ohmsLaw.csv	CSV file	Used to store current, voltage and resistance (user's and calculated values).
meterBridge.csv	CSV file	Used to store resistance and length values (user's and calculated values).
meterBridgeSnP.csv	CSV file	Used to store resistance and length values (user's and calculated values).

galvanometer.csv	CSV file	Used to store values of number of divisions and resistance (user's and calculated values).
convexMirror.csv	CSV file	Used to store readings of position of object, lens, mirror and screen.
diode.csv	CSV file	Used to store voltmeter and ammeter readings as observed.
VolmFAS.txt	Text file	Contains content for 'Estimations of Potassium Permanganate using FAS' experiment
VolmOxalic.txt	Text file	Contains content for 'Estimations of Potassium Permanganate using Oxalic Acid' experiment
rate_temp.txt	Text file	Contains content for 'Rate of Reaction – Effect of Temperature' experiment
rate_conc.txt	Text file	Contains content for 'Rate of Reaction – Effect of Concentration' experiment
concave mirror.txt	Text file	Contains content for 'Concave Mirror' experiment
convex lens.txt	Text file	Contains content for 'Convex Lens' experiment
ohms law.txt	Text file	Contains content for 'Ohms Law' experiment
meter bridge.txt	Text file	Contains content for 'Meter Bridge' experiment

meter bridge snp.txt	Text file	Contains content for 'Meter Bridge – Series and Parallel' experiment
galvanometer.txt	Text file	Contains content for 'Galvanometer' experiment
convex mirror.txt	Text file	Contains content for 'Convex Mirror' experiment
diode.txt	Text file	Contains content for 'Diode Characteristics' experiment

Contents of Datasets

Physics:

concave mirror.txt:

AIM:

To find the value of image distance 'v' for different object distances 'u' for the given concave mirror and hence find the focal length 'f' from calculation and by plotting a graph between u & v and 1/u & 1/v.

APPARATUS REQUIRED:

Illuminated wire gauze (object), concave mirror, screen, mirror holder and a metre scale.

FORMULA:

Using sign convention, 1/v + 1/u = 1/f

u - Object distance

v - Image distance

f - Focal length

PROCEDURE:

The given concave mirror is mounted on the holder and focused at a distant object. A sharp image is obtained on the screen. The distance between the mirror and the screen gives the rough focal length of the mirror. The mirror is then placed in front of the illuminated wire gauze. A well-defined image is formed on the screen. The position of wire gauze, mirror and screen are noted. The screen position is located for four different positions of object, which are between f and 2f and four positions beyond 2f. Observations are tabulated. Focal length is calculated for each observation using mirror formula and average value of 'f' is found.

U-V GRAPH:

A graph is drawn between u and v with an equal kink and an equal scale. The curve is a rectangular hyperbola. The angle bisector of the angle at the origin meets the curve at (2f, 2f). Hence 'f' is determined.

1/u - 1/v GRAPH:

A graph between 1/u and 1/v is a straight line with negative slope. The graph is drawn with equal kink and equal scale. It is extrapolated to cut the two axes. The point of intersection of graph on two axes, gives value of 1/f if the origin is (0,0). If a kink is chosen then the kink on y-axis is added to the x-intercept to find 1/f. Similarly the kink on x-axis is added to the y-intercept to find 1/f.

Chemistry:

VolmFAS.txt:

AIM:

To determine the molarity of the given Potassium Permanganate solution, using the standard FAS solution.

APPARATUS:

Burette, pipette, burette stand, conical flask, funnel porcelain tile, test tube, standard flask (100 ml), weighing balance.

CHEMICALS REQUIRED:

Potassium permanganate solution, ferrous ammonium sulphate crystals, dilute sulphuric acid, distilled water.

THEORY:

Potassium permanganate is a strong oxidising agent especially in acidic medium while ferrous ammonium sulphate is a reducing agent in the reaction. 5 moles a ferrous sulphate are oxidized by 1 mole of potassium permanganate. Here only ferrous sulphate is used up and ammonium sulphate is an inert electrolyte, which prevents air oxidation of Fe" to Fe" in the crystalline state.

PROCEDURE:

A standard solution (100 ml) of ferrous ammonium sulphate was prepared by weighed amount of salt and making it upto 100 ml in a standard flask. The burette was washed with tap water, distilled water and then with a few ml. of the given potassium permanganate solution. After clamping it to the stand, it was filled with the given potassium permanganate solution taking care to remove all air bubbles in any part of the burette including the nozzle. The pipette was rinsed with tap water, distilled water and then with a few ml of ferrous ammonium sulphate solution. 20ml of the solution was pipetted out in to a clean, dry conical flask and a test tube full of dilute sulphuric acid was added and titrated against potassium permanganate taken in the burette. Each drop of permanganate solution was added after the previous one was decolourised. Titration was continued till the appearance of permanent pale pink colour and was repeated for concordant values. The results were tabulated.

List of Global Variables and Functions

Global Variables

Global Variable	Purpose
root	Main tkinter window
frame	A frame in root that contains all other widgets.

User Defined Functions

User Defined Functions	Purpose
chemexpts	For accessing list of chemistry experiments.
phyexpts	For accessing list physics experiments.
VolmFAS	Command for 'Estimations of Potassium Permanganate using FAS' button.
VolmOxalic	Command for 'Estimations of Potassium Permanganate using Oxalic Acid' button.
rate_temp	Command for 'Rate of Reaction – Effect of Temperature' button.
rate_con	Command for 'Rate of Reaction – Effect of Concentration' button.
concaveMirror1	Command for 'Concave Mirror' button.
convexLens2	Command for 'Convex Lens' button.
ohmsLaw3	Command for 'Ohm's Law' button.
meterBridge4	Command for 'Meter Bridge' button.
meterBridgeSnP5	Command for 'Meter Bridge – Series and Parallel button.
galvanometer6	Command for 'Galvanometer' button.
convexMirror7	Command for 'Convex Mirror' button.
diode8	Command for 'Diode Characteristics' button.

save/save1/save2/save3	Nested function for saving all the values entered by the user in the file. Command for ‘Save’ button.
result	Nested function for calculations/plotting graphs using values saved in the file and displaying the final result. Command for ‘Show Result’ button

Source Code

```
from tkinter import * #imports all functions and built-in modules present in tkinter library
from tkinter import ttk #to style tkinter widgets
from tkinter import messagebox #to display message boxes
import matplotlib.pyplot as plt #to display graphs
import statistics as stat #for calculations
import csv #file handling

#Command for 'Estimations of Potassium Permanganate using FAS' button
def VolmFAS():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f = open('VolmFAS.txt')
    l = f.readlines()
    f.close()

    heading = Label(second_frame ,text= 'Estimation of Potassium
Permanganate using FAS\n', font =('Times New Roman',25,'bold'),
bg='white', width= 80)
```

```
    heading.grid(row=0, column=0, sticky= 'n')

    aim_subH= Label(second_frame ,text=l[0], font =('Times New
Roman',20,'bold'), bg= 'white')
    aim_subH.grid(row=1, column=0, sticky= 'w')

    aim= Label(second_frame , text= l[1], font= ('Times New
Roman',20), justify= 'left', bg= 'white')
    aim.grid(row= 2, column=0, sticky= 'w')

    apparatus_subH= Label(second_frame ,text=l[2], font =('Times
New Roman',20,'bold'), bg= 'white')
    apparatus_subH.grid(row= 3, column=0, sticky= 'w')

    apparatus= Label(second_frame , text= l[3], font= ('Times New
Roman', 20), justify= 'left', bg='white')
    apparatus.grid(row=4, column=0, sticky= 'w')

    chemicals_subH= Label(second_frame , text=l[4], font =('Times
New Roman',20,'bold'), bg= 'white')
    chemicals_subH.grid(row=5, column= 0, sticky= 'w')

    chemicals= Label(second_frame , text=l[5], font= ('Times New
Roman',20), justify= 'left', bg= 'white')
    chemicals.grid(row= 6, column=0, sticky= 'w')

    theory_subH= Label(second_frame , text=l[6], font =('Times New
Roman',20,'bold'), bg= 'white')
    theory_subH.grid(row=7, column= 0, sticky= 'w')

    theory= Label(second_frame , text=l[7], font= ('Times New
Roman',20), justify= 'left', bg= 'white',wraplength=1300)
    theory.grid(row=8, column= 0, sticky= 'w')
```

```
    procedure_subH= Label(second_frame , text=l[8], font =('Times
New Roman',20,'bold'), bg= 'white')
    procedure_subH.grid(row=9, column= 0, sticky= 'w')

    procedure= Label(second_frame , text=l[9], font= ('Times New
Roman',20), justify= 'left', bg= 'white',wraplength=1300)
    procedure.grid(row=10, column= 0, sticky= 'w')

    tpp= Label(second_frame , text='''
                    TEN-POINT PROCEDURE

        +-----------------------------------------------+
        |Burette Solution           |KMnO4              |
        |___________________________|___________________|
        |                           |                   |
        |Pippete Solution           |FAS                |
        |___________________________|___________________|
        |                           |                   |
        |Indicator                  |Self               |
        |___________________________|___________________|
        |                           |                   |
        |End point                  |Colourless to Pink |
        |___________________________|___________________|
        |                           |                   |
        |Molar Mass of FAS          |392g/mol           |
        |___________________________|___________________|
        |                           |                   |
        |Molar Mass of KMnO4        |158g/mol           |
        |___________________________|___________________|
        |                           |                   |
        |No. of e transfered (KMnO4)|5e                 |
        |___________________________|___________________|
        |                           |                   |
        |No. of e transfered (FAS)  |1e                 |
        |___________________________|___________________|
```

```
        |                                |                    |
        |Temperature                     |Room Temperature    |
        |________________________________|____________________|
        |                                |                    |
        |Medium                          |Acidic              |
        +------------------------------------------------------+
''', font= ('Courier',17), justify= 'left', bg= 'white')
    tpp.grid(row=11, column= 0, sticky= 'w')

    precautions_subH= Label(second_frame, text= 'PRECAUTIONS:',
font= ('Times New Roman',20, 'bold'), justify= 'left', bg=
'white')
    precautions_subH.grid(row=12, column= 0, sticky= 'w')

    precautions= Label(second_frame, text=
'''□   All apparatus must be thoroughly washed with distilled
water.
□   Only the burette and pipette should be raised with their
respective solutions.
□   Upper meniscus reading should be taken for KMnO, and lower one
for FAS.
□   There should be no air bubbles in the burette and its nozzle.
□   Each drop of KMnO, should be decolourised before the next is
added.
''', font= ('Times New Roman',20), justify= 'left', bg= 'white')
    precautions.grid(row=13, column= 0, sticky= 'w')

    observations_subH= Label(second_frame, text= 'OBSERVATION:',
font =('Times New Roman',20,'bold'), bg= 'white')
    observations_subH.grid(row=14, column=0, sticky= 'w')

    file = open('VolmFAS.csv','w+',newline='')
    w = csv.writer(file)
    r = csv.reader(file)
    w.writerow(['Molarity of FAS','Concordant Value'])
```

```
    entry1 = Label(second_frame,text='Molarity of FAS:
',bg='white',font=('Times New
Roman',20),justify='left').grid(row=15,column=0,sticky='w')
    entry2 = Label(second_frame,text='Concordent Burette Reading:
',bg='white',font=('Times New
Roman',20),justify='left').grid(row=16,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row= 15, column= 0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row= 16, column= 0)


    def save():
        Molarity = float(input1.get())
        Volume = float(input2.get())
        l = [Molarity,Volume]
        w.writerow(l)
        messagebox.showinfo("Reading","Saved succesfully")

    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        Molarity = float(readings[0][0])
        Volume = float(readings[0][1])
        FAS_mass= round(((Molarity*392)*0.1), 3)
        M_KMnO= round(((Molarity*20*1)/(Volume*5)), 5)
        Str_KMnO=  round(M_KMnO*158, 4)

        label1 = Label(second_frame,text='Mass of FAS =
'+str(FAS_mass)+' g',bg='white',font=('Times New
Roman',20)).grid(row= 18, column= 0)
```

```
        label2 = Label(second_frame,text='Molarity of KMnO4 =
'+str(M_KMnO)+' M',bg='white',font=('Times New
Roman',20)).grid(row= 19, column= 0)
        label2 = Label(second_frame,text='Strength of KMnO4 =
'+str(Str_KMnO)+' M',bg='white',font=('Times New
Roman',20)).grid(row= 20, column= 0)

    Save = Button(second_frame,text='Save',width=30,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=21,column=0,sticky='w')

    result = Button(second_frame,text='Show
result',width=30,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    result.grid(row=22,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff', command=
chemexpts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))
```

#Command for 'Estimations of Potassium Permanganate using Oxalic Acid' button

```
def VolmOxalic():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f = open('VolmOxalic.txt')
    l = f.readlines()
    f.close()

    heading = Label(second_frame,text='Estimation of Potassium
Permanganate using Oxalic Acid\n', font =('Times New
Roman',25,'bold'), width= 80, bg= 'white')
    heading.grid(row=0, column=0, sticky= 'n')

    aim_subH= Label(second_frame,text=l[0], font =('Times New
Roman',20,'bold'), bg= 'white', justify= 'left')
    aim_subH.grid(row=1, column= 0, sticky= 'w')

    aim= Label(second_frame, text=l[1], font= ('Times New
Roman',20), justify= 'left', bg= 'white',wraplength = 1300)
    aim.grid(row=2, column= 0, sticky= 'w')

    apparatus_subH= Label(second_frame,text=l[2], font =('Times
New Roman',20,'bold'), bg= 'white')
    apparatus_subH.grid(row=3, column=0, sticky= 'w')
```

```
    apparatus= Label(second_frame, text= l[3], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    apparatus.grid(row= 4, column=0, sticky= 'w')

    chemicals_subH= Label(second_frame, text= l[4], font =('Times
New Roman',20,'bold'), bg= 'white')
    chemicals_subH.grid(row=5, column=0, sticky= 'w')

    chemicals= Label(second_frame, text=l[5], font= ('Times New
Roman', 20), bg= 'white', justify= 'left',wraplength=1300)
    chemicals.grid(row=6, column=0, sticky= 'w')

    theory_subH= Label(second_frame, text= l[6], font =('Times New
Roman',20,'bold'), bg= 'white')
    theory_subH.grid(row=7, column=0, sticky= 'w')

    theory= Label(second_frame, text=l[7], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    theory.grid(row=8, column=0, sticky= 'w')

    procedure_subH= Label(second_frame, text=l[8], font =('Times
New Roman',20,'bold'), bg= 'white')
    procedure_subH.grid(row=9, column=0, sticky= 'w')

    procedure= Label(second_frame, text=l[9], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    procedure.grid(row=10, column=0, sticky= 'w')

    tpp= Label(second_frame, text='''
                          TEN-POINT PROCEDURE

        +-----------------------------------------------------+
        |Burette Solution                   |KMnO4            |
        |___________________________________|_________________|
        |                                   |                 |
```

```
        |Pippete Solution                    |Oxalic Acid        |
        |____________________________________|___________________|
        |                                    |                   |
        |Indicator                           |Self               |
        |____________________________________|___________________|
        |                                    |                   |
        |End point                           |Colourless to Pink |
        |____________________________________|___________________|
        |                                    |                   |
        |Molar Mass of oxalic acid           |126g/mol           |
        |____________________________________|___________________|
        |                                    |                   |
        |Molar Mass of KMnO4                 |158g/mol           |
        |____________________________________|___________________|
        |                                    |                   |
        |No. of e transfered (KMnO4)         |5e                 |
        |____________________________________|___________________|
        |                                    |                   |
        |No. of e transfered (oxalic acid)   |2e                 |
        |____________________________________|___________________|
        |                                    |                   |
        |Temperature                         |60°C to 70°C       |
        |____________________________________|___________________|
        |                                    |                   |
        |Medium                              |Acidic             |
        +----------------------------------------------------------+
''', font= ('courier', 17), justify= 'left', bg= 'white')
    tpp.grid(row=11, column=0, sticky= 'w')

    precautions_subH= Label(second_frame, text= 'PRECAUTIONS:',
font= ('Times New Roman',20, 'bold'), justify= 'left', bg=
'white')
    precautions_subH.grid(row=12, column= 0, sticky= 'w')
```

```
    precautions= Label(second_frame, text='''   All apparatus
must be thoroughly washed.
   Only the burette and pipette should be rinsed with their
respective solutions.
   Upper meniscus reading should be taken for KMnO, and lower one
for Oxalic Acid.
   There should be no air bubbles in the burette's nozzle.
   Each drop of potassium permanganate added should be
decolourised before the next one is added.
''', font= ('Times New Roman',20), justify= 'left', bg= 'white')
    precautions.grid(row=13, column= 0, sticky= 'w')

    observations_subH= Label(second_frame, text= 'OBSERVATION:',
font =('Times New Roman',20,'bold'), bg= 'white')
    observations_subH.grid(row=14, column=0, sticky= 'w')

    file = open('VolmOxalic.csv','w+',newline='')
    w = csv.writer(file)
    r = csv.reader(file)
    w.writerow(['Molarity of Oxalic','Concordent Value'])

    entry1 = Label(second_frame,text='Molarity of Oxalic:
',bg='white',font=('Times New
Roman',20),justify='left').grid(row=15,column=0,sticky='w')
    entry2 = Label(second_frame,text='Concordent Burette Reading:
',bg='white',font=('Times New
Roman',20),justify='left').grid(row=16,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row= 15, column= 0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row= 16, column= 0)
```

```
    def save():
        Molarity = float(input1.get())
        Volume = float(input2.get())
        l = [Molarity,Volume]
        w.writerow(l)
        messagebox.showinfo("Reading","Saved succesfully")

    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        Molarity = float(readings[0][0])
        Volume = float(readings[0][1])
        Oxalic_mass= round(((Molarity*392)*0.1), 3)
        M_KMnO= round(((Molarity*20*1)/(Volume*5)), 5)
        Str_KMnO=  round(M_KMnO*158, 4)

        label1 = Label(second_frame,text='Mass of Oxalic =
'+str(Oxalic_mass)+' g',bg='white',font=('Times New
Roman',20)).grid(row= 18, column= 0)
        label2 = Label(second_frame,text='Molarity of KMnO4 =
'+str(M_KMnO)+' M',bg='white',font=('Times New
Roman',20)).grid(row= 19, column= 0)
        label2 = Label(second_frame,text='Strength of KMnO4 =
'+str(Str_KMnO)+' M',bg='white',font=('Times New
Roman',20)).grid(row= 20, column= 0)

    Save = Button(second_frame,text='Save',width=30,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=21,column=0,sticky='w')

    result = Button(second_frame,text='Show
result',width=30,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
```

```
    result.grid(row=22,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))
```

#Command for 'Rate of Reaction – Effect of Temperature' button

```
def rate_temp():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f = open('rate_temp.txt')
    l = f.readlines()
```

```
    f.close()

    heading = Label(second_frame,text='Rate Of Reaction- Effect of
Temperature\n', font =('Times New Roman',25,'bold'), width= 80,
bg= 'white')
    heading.grid(row=0, column=0, sticky= 'n')

    aim_subH= Label(second_frame,text=l[0], font =('Times New
Roman',20,'bold'), bg= 'white', justify= 'left')
    aim_subH.grid(row=1, column= 0, sticky= 'w')

    aim= Label(second_frame, text= l[1], font= ('Times New
Roman',20), justify= 'left', bg= 'white')
    aim.grid(row=2, column= 0, sticky= 'w')

    apparatus_subH= Label(second_frame,text=l[2], font =('Times
New Roman',20,'bold'), bg= 'white')
    apparatus_subH.grid(row=3, column=0, sticky= 'w')

    apparatus= Label(second_frame, text= l[3], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    apparatus.grid(row= 4, column=0, sticky= 'w')

    chemicals_subH= Label(second_frame, text= l[4], font =('Times
New Roman',20,'bold'), bg= 'white')
    chemicals_subH.grid(row=5, column=0, sticky= 'w')

    chemicals= Label(second_frame, text=l[5], font= ('Times New
Roman', 20), bg= 'white', justify= 'left',wraplength=1300)
    chemicals.grid(row=6, column=0, sticky= 'w')

    theory_subH= Label(second_frame, text= l[6], font =('Times New
Roman',20,'bold'), bg= 'white')
    theory_subH.grid(row=7, column=0, sticky= 'w')
```

```
    theory= Label(second_frame, text=l[7], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    theory.grid(row=8, column=0, sticky= 'w')

    procedure_subH= Label(second_frame, text=l[8], font =('Times
New Roman',20,'bold'), bg= 'white')
    procedure_subH.grid(row=9, column=0, sticky= 'w')

    procedure= Label(second_frame, text=l[9], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    procedure.grid(row=10, column=0, sticky= 'w')

    precautions_subH= Label(second_frame, text= 'PRECAUTIONS:',
font= ('Times New Roman',20, 'bold'), justify= 'left', bg=
'white')
    precautions_subH.grid(row=11, column= 0, sticky= 'w')

    precautions= Label(second_frame, text=
'''□   The stopclock should be stopped as soon as turbidity
appears.
□   Concentrated nitric acid should be used to clean the test
tube.
□   The room temperature should be noted.
''', font= ('Times New Roman',20), justify= 'left', bg= 'white')
    precautions.grid(row=12, column= 0, sticky= 'w')

    observations_subH= Label(second_frame, text= 'OBSERVATION:',
font =('Times New Roman',20,'bold'), bg= 'white')
    observations_subH.grid(row=13, column=0, sticky= 'w')

    text= Label(second_frame, text=
''' Please enter the time take for turbidity to appear at each
Temperature:''', font= ('Times New Roman', 20), justify= 'left',
bg= 'white')
    text.grid(row=14, column=0, sticky= 'w')
```

```
    file = open('ratc.csv','w+',newline='')
    w = csv.writer(file)
    r = csv.reader(file)
    w.writerow(['Temperature', 'Time'])

    entry1 = Label(second_frame,text='10 Degree celcius above
temperature: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=15,column=0,sticky='w')
    entry2 = Label(second_frame,text='20 Degree celcius above
temperature: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=16,column=0,sticky='w')
    entry3 = Label(second_frame,text='30 Degree celcius above
temperature: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=17,column=0,sticky='w')
    entry4 = Label(second_frame,text='40 Degree celcius above
temperature: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=18,column=0,sticky='w')
    entry5 = Label(second_frame,text='50 Degree celcius above
temperature: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=19,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row= 15, column= 0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row= 16, column= 0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input3.grid(row= 17, column= 0)
    input4 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input4.grid(row= 18, column= 0)
    input5 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
```

```
    input5.grid(row= 19, column= 0)

    def save():
        t1 = float(input1.get())
        t2 = float(input2.get())
        t3 = float(input3.get())
        t4 = float(input4.get())
        t5 = float(input5.get())
        l = [[+10, t1], [+20, t2], [+30, t3], [+40, t4], [+50,
t5]]
        w.writerows(l)
        messagebox.showinfo("Reading","Saved succesfully")

    def result():
        file.seek(0)
        readings = list(r)
        x=[]
        y=[]
        for i in readings:
            x+=[i[0]]
            y+=[i[1]]
        plt.plot(x, y)
        plt.xlabel('Volume of Sodium Thiosulphate')
        plt.ylabel('Time Taken for Turbidity')
        plt.title('Temperature of Surrounding -vs- Time Taken for
Turbidity')
        plt.show()

    Save = Button(second_frame,text='Save',width=30,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=20,column=0,sticky='w')
```

```
    result = Button(second_frame,text='Show
result',width=30,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    result.grid(row=21,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))
```

#Command for 'Rate of Reaction – Effect of Concentration' button

```
def rate_con():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')
```

```
    f = open('rate_conc.txt')
    l = f.readlines()
    f.close()

    heading = Label(second_frame,text='Rate Of Reaction- Effect of
Concentration\n', font =('Times New Roman',25,'bold'), width= 80,
bg= 'white')
    heading.grid(row=0, column=0, sticky= 'n')

    aim_subH= Label(second_frame,text=l[0], font =('Times New
Roman',20,'bold'), bg= 'white', justify= 'left')
    aim_subH.grid(row=1, column= 0, sticky= 'w')

    aim= Label(second_frame, text= l[1], font= ('Times New
Roman',20), justify= 'left', bg= 'white',wraplength=1300)
    aim.grid(row=2, column= 0, sticky= 'w')

    apparatus_subH= Label(second_frame,text=l[2], font =('Times
New Roman',20,'bold'), bg= 'white')
    apparatus_subH.grid(row=3, column=0, sticky= 'w')

    apparatus= Label(second_frame, text= l[3], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    apparatus.grid(row= 4, column=0, sticky= 'w')

    chemicals_subH= Label(second_frame, text= l[4], font =('Times
New Roman',20,'bold'), bg= 'white')
    chemicals_subH.grid(row=5, column=0, sticky= 'w')

    chemicals= Label(second_frame, text=l[5], font= ('Times New
Roman', 20), bg= 'white', justify= 'left')
    chemicals.grid(row=6, column=0, sticky= 'w')
```

```
    theory_subH= Label(second_frame, text= l[6], font =('Times New
Roman',20,'bold'), bg= 'white')
    theory_subH.grid(row=7, column=0, sticky= 'w')

    theory= Label(second_frame, text=l[7], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    theory.grid(row=8, column=0, sticky= 'w')

    procedure_subH= Label(second_frame, text=l[8], font =('Times
New Roman',20,'bold'), bg= 'white')
    procedure_subH.grid(row=9, column=0, sticky= 'w')

    procedure= Label(second_frame, text=l[9], font= ('Times New
Roman', 20), justify= 'left', bg= 'white',wraplength=1300)
    procedure.grid(row=10, column=0, sticky= 'w')

    precautions_subH= Label(second_frame, text='PRECAUTIONS: ',
font= ('Times New Roman',20, 'bold'), justify= 'left', bg=
'white')
    precautions_subH.grid(row=11, column= 0, sticky= 'w')

    precautions= Label(second_frame,text='''□   Flask and tile
should be cleaned thoroughly.
□   The stop clock should be started immediately after the
addition of hydrochloric acid.
□   Concentrated nitric acid should be used to clean the flask
thoroughly after each reading.
□   Stock clock shuld be stopped as soon as turbidity appears.
''', font= ('Times New Roman',20), justify= 'left', bg= 'white')
    precautions.grid(row=12, column= 0, sticky= 'w')

    observations_subH= Label(second_frame, text= 'OBSERVATIONS: ',
font =('Times New Roman',20,'bold'), bg= 'white')
    observations_subH.grid(row=13, column=0, sticky= 'w')
```

```
    text= Label(second_frame, text=
''' Please enter the time take for turbidity to appear for each
concentration:''', font= ('Times New Roman', 20), justify= 'left',
bg= 'white')
    text.grid(row=14, column=0, sticky= 'w')

    file = open('ratc.csv','w+',newline='')
    w = csv.writer(file)
    r = csv.reader(file)
    w.writerow(['Volume of Sodium Thiosulphate', 'Volume of
Water', 'Time'])

    entry1 = Label(second_frame,text='Sodium Thiosulphate- 25 ml,
Water- 0 ml: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=15,column=0,sticky='w')
    entry2 = Label(second_frame,text='Sodium Thiosulphate- 20 ml,
Water- 5 ml: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=16,column=0,sticky='w')
    entry3 = Label(second_frame,text='Sodium Thiosulphate- 15 ml,
Water- 10 ml: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=17,column=0,sticky='w')
    entry4 = Label(second_frame,text='Sodium Thiosulphate- 10 ml,
Water- 15 ml: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=18,column=0,sticky='w')
    entry5 = Label(second_frame,text='Sodium Thiosulphate- 5 ml,
Water- 20 ml: ',bg='white',font=('Times New
Roman',20),justify='left').grid(row=19,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row= 15, column= 0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row= 16, column= 0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
```

```
    input3.grid(row= 17, column= 0)
    input4 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input4.grid(row= 18, column= 0)
    input5 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input5.grid(row= 19, column= 0)

    def save():
        t1 = float(input1.get())
        t2 = float(input2.get())
        t3 = float(input3.get())
        t4 = float(input4.get())
        t5 = float(input5.get())
        l = [[25, 0, t1], [20, 5, t2], [15, 10, t3], [10, 15, t4],
[5, 20, t5]]
        w.writerows(l)
        messagebox.showinfo("Reading","Saved succesfully")

    def result():
        file.seek(0)
        readings = list(r)
        x=[]
        y=[]
        for i in readings:
            x+=[i[2]]
            y+=[i[0]]
        plt.plot(x, y)
        plt.xlabel('Volume of Sodium Thiosulphate')
        plt.ylabel('Time Taken for Turbidity')
        plt.title('Volume of Sodium Thiosulphate -vs- Time Taken
for Turbidity')
        plt.show()
```

```
    Save = Button(second_frame,text='Save',width=30,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=20,column=0,sticky='w')

    result = Button(second_frame,text='Show
result',width=30,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    result.grid(row=21,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))

#For accessing list of chemistry experiments
def chemexpts():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()

    heading = Label(frame,text='List of Chemistry Experiments',
font =('Times New Roman',25,'bold'),bg='white')
```

```
    heading.place(x=580,y=30)

    volumetricFASbutton = Button(frame,text='Estimation of
Potassium Permanganate using FAS',bg='#00FFFF',width=80, font
=('Times New Roman',20,'bold'), command= VolmFAS)
    volumetricFASbutton.place(x=140,y=100)

    volumetricOXALICbutton = Button(frame,text='Estimation of
Potassium Permanganate using Oxalic Acid',bg='#E0FFFF',width=80,
font =('Times New Roman',20,'bold'), command= VolmOxalic)
    volumetricOXALICbutton.place(x=140, y=200)

    rate_tempbutton = Button(frame,text='Rate of Reaction- Effect
of Temperature',bg='#00FFFF',width=80, font =('Times New
Roman',20,'bold'), command= rate_temp)
    rate_tempbutton.place(x=140, y=300)

    rate_conbutton = Button(frame,text='Rate of Reaction- Effect
of Concentration',bg='#E0FFFF',width=80, font =('Times New
Roman',20,'bold'), command= rate_con)
    rate_conbutton.place(x=140, y=400)

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)

#Command for 'Concave Mirror' button
def concaveMirror1():
    global root
    global frame
```

```
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f = open('concave mirror.txt')
    l= f.readlines()
    f.close()

    title = Label(second_frame,text='Concave Mirror',font=('Times
New Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    aim = Label(second_frame,text= l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left', wraplength=1500)
    aim.grid(row=1,column=0,sticky='w')

    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left')
    matreqd.grid(row=2,column=0,sticky='w')

    formula =
Label(second_frame,text=l[4]+l[5]+l[6]+l[7]+l[8],font=('Times New
Roman',20),bg='white',justify='left')
    formula.grid(row=3,column=0,sticky='w')

    procedure = Label(second_frame,text=l[9]+l[10],font=('Times
New Roman',20),bg='white',justify='left', wraplength=1500)
    procedure.grid(row=4,column=0,sticky='w')
```

```
    uvgraph = Label(second_frame,text=l[11]+l[12],font=('Times New
Roman',20),bg='white',justify='left', wraplength=1500)
    uvgraph.grid(row=5,column=0,sticky='w')

    reciprocalgraph =
Label(second_frame,text=l[13]+l[14],font=('Times New
Roman',20),bg='white',justify='left', wraplength=1500)
    reciprocalgraph.grid(row=6,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The mirror, source and screen should be at the same height.
• The principal axis of the mirror should be parallel to the
surface of the table. The horizontal and vertical
   cross wires should be equally clear at image positions.
• The screen should be placed as close to the axis of the mirror
as possible.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    precautions.grid(row=7,column=0,sticky='w')

    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• Spherical aberration of the mirror
• Parallax error while taking reading\n''',font=('Times New
Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=8,column=0,sticky='w')

    file = open('concaveMirror.csv','w+',newline='')
    r = csv.reader(file)
    w = csv.writer(file)
    w.writerow(['Position of object (cm)','Position of mirror
(cm)','Position of screen (cm)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
```

```
    obs.grid(row=9,column=0,sticky='w')

    entry1 = Label(second_frame,text='Position of object
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=10,column=0,sticky='w')
    entry2 = Label(second_frame,text='Position of mirror
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=11,column=0,sticky='w')
    entry3 = Label(second_frame,text='Position of screen
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=12,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row=10,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=11,column=0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input3.grid(row=12,column=0)

    def save():
        tempvar1 = float(input1.get())
        tempvar2 = float(input2.get())
        tempvar3 = float(input3.get())
        l = [tempvar1,tempvar2,tempvar3]
        w.writerow(l)
        messagebox.showinfo("Reading","Saved succesfully")

    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        one_u = []
        one_v = []
```

```
        one_f = []
        for i in readings:
            x = float(i[0])
            y = float(i[2])
            x1 = round((1/x),5)
            y1 = round((1/y),5)
            one_u.append(x1)
            one_v.append(y1)
            one_f.append(x1+y1)
        mean = stat.mean(one_f)
        f = round(1/mean,1)
        label1 = Label(second_frame,text='Mean value of 1/f =
'+str(mean)+' cm⁻¹',bg='white',font=('Times New
Roman',20)).place(x=500,y=1450)
        label2 = Label(second_frame,text='Mean value of f =
'+str(f)+' cm',bg='white',font=('Times New
Roman',20)).place(x=500,y=1500)

    Save = Button(second_frame,text='Save',width=30,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=13,column=0,sticky='w')
    res = Button(second_frame,text='Show
result',width=30,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    res.grid(row=14,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
```

```
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))

#Command for 'Convex Lens' button
def convexLens2():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f = open('convex lens.txt')
    l = f.readlines()
    f.close()

    title = Label(second_frame,text='Convex Lens',font=('Times New
Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    aim = Label(second_frame,text=l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    aim.grid(row=1,column=0,sticky='w')
```

```
    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    matreqd.grid(row=2,column=0,sticky='w')

    formula =
Label(second_frame,text=l[4]+l[5]+l[6]+l[7]+l[8]+l[9],font=('Times
New Roman',20),bg='white',justify='left',wraplength=1500)
    formula.grid(row=3,column=0,sticky='w')


    procedure = Label(second_frame,text=l[10]+l[11],font=('Times
New Roman',20),bg='white',justify='left',wraplength=1500)
    procedure.grid(row=4,column=0,sticky='w')

    uvgraph = Label(second_frame,text=l[12]+l[13],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    uvgraph.grid(row=5,column=0,sticky='w')

    reciprocalgraph =
Label(second_frame,text=l[14]+l[15],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    reciprocalgraph.grid(row=6,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The lens, source and screen should be at the same height.
• The horizontal and vertical cross wires must be clearly visible
on the screen.
• The lens, source and screen should be at the same
line.\n''',font=('Times New Roman',20),bg='white',justify='left')
    precautions.grid(row=7,column=0,sticky='w')

    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• Spherical aberration of the lens.
• Parallax error while taking reading.
• Thickness of the lens may not be uniform.
```

```
• The image may not be formed properly.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=8,column=0,sticky='w')

    file = open('convexLens.csv','w+',newline='')
    r = csv.reader(file)
    w = csv.writer(file)
    w.writerow(['Position of object (cm)','Position of lens
(cm)','Position of screen (cm)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
    obs.grid(row=9,column=0,sticky='w')

    entry1 = Label(second_frame,text='Position of object
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=10,column=0,sticky='w')
    entry2 = Label(second_frame,text='Position of lens
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=11,column=0,sticky='w')
    entry3 = Label(second_frame,text='Position of screen
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=12,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row=10,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=11,column=0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input3.grid(row=12,column=0)

    def save():
```

```
        tempvar1 = float(input1.get())
        tempvar2 = float(input2.get())
        tempvar3 = float(input3.get())
        l = [tempvar1,tempvar2,tempvar3]
        w.writerow(l)
        messagebox.showinfo("Reading","Saved succesfully")

    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        one_u = []
        one_v = []
        one_f = []
        for i in readings:
            x = float(i[1])
            y = float(i[2])-float(i[1])
            x1 = round((1/x),5)
            y1 = round((1/y),5)
            one_u.append(x1)
            one_v.append(y1)
            one_f.append(x1+y1)
        mean = stat.mean(one_f)
        f = round(1/mean,1)
        label1 = Label(second_frame,text='Mean value of 1/f =
'+str(mean)+' cm⁻¹',bg='white',font=('Times New
Roman',20)).place(x=500,y=1530)
        label2 = Label(second_frame,text='Mean value of f =
'+str(f)+' cm',bg='white',font=('Times New
Roman',20)).place(x=500,y=1580)

    Save = Button(second_frame,text='Save',width=30,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=13,column=0,sticky='w')
```

```
    res = Button(second_frame,text='Show
result',width=30,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    res.grid(row=14,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))

#Command for 'Ohm's Law' button
def ohmsLaw3():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')
```

```
    f = open('ohms law.txt')
    l = f.readlines()
    f.close()

    title = Label(second_frame,text='Ohm\'s Law',font=('Times New
Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    aim = Label(second_frame,text=l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    aim.grid(row=1,column=0,sticky='w')

    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    matreqd.grid(row=2,column=0,sticky='w')

    formula =
Label(second_frame,text=l[4]+l[5]+l[6]+l[7]+l[8]+l[9]+l[10]+l[11]+
l[12],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    formula.grid(row=3,column=0,sticky='w')

    procedure =
Label(second_frame,text=l[13]+l[14]+l[15]+l[16]+l[17]+l[18]+l[19]+
l[20]+l[21]+l[22]+l[23],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    procedure.grid(row=4,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The connections should be neat, clean and tight.
• Thick copper wires should be used for connections.
• Ammeter and voltmeter should be of proper range.
• Do not pass large currents as the resistor may get heated up.
```

```
• The connections to the rheostat must be made by using one
terminal at the top and one at the bottom.
• Ensure that current enters the ammeter and the voltmeter through
their positive terminals.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    precautions.grid(row=5,column=0,sticky='w')

    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• Rheostat may have high resistance.
• The instrument screws may be loose.
• Thick connecting wires may not be available.\n''',font=('Times
New Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=6,column=0,sticky='w')

    file = open('ohmsLaw.csv','w+',newline='')
    r = csv.reader(file)
    w = csv.writer(file)
    w.writerow(['Current (A)','Voltage (V)','Resistance (ohm)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
    obs.grid(row=7,column=0,sticky='w')

    length = Label(second_frame,text='Length of coil
(m):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=8,column=0,sticky='w')
    radius = Label(second_frame,text='Radius of coil
(m):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=9,column=0,sticky='w')
    entry1 = Label(second_frame,text='Current
(A):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=10,column=0,sticky='w')
```

```
    entry2 = Label(second_frame,text='Voltage
(V):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=11,column=0,sticky='w')
    lenn = Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    lenn.grid(row=8,column=0)
    rad = Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    rad.grid(row=9,column=0)
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row=10,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=11,column=0)

    def save():
        tempvar1 = float(input1.get())
        tempvar2 = float(input2.get())
        l = [tempvar1,tempvar2,round(tempvar2/tempvar1,2)]
        w.writerow(l)
        messagebox.showinfo("Reading","Saved succesfully")

    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        resis = []
        for i in readings:
            resis.append(float(i[2]))
        mean = stat.mean(resis)
        resistivity =
round((mean*3.14*(float(rad.get())**2))/float(lenn.get()),10)
        label1 = Label(second_frame,text='Mean value of resistance
= '+str(mean)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=500,y=1550)
```

```
        label2 = Label(second_frame,text='Resistivity =
'+str(resistivity)+' \u2126 m',bg='white',font=('Times New
Roman',20)).place(x=500,y=1600)

    Save = Button(second_frame,text='Save',width=30,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=13,column=0,sticky='w')

    res = Button(second_frame,text='Show
result',width=30,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    res.grid(row=14,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',
command=chemexpts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))
```

#Command for 'Meter Bridge' button

```
def meterBridge4():
    global root
    global frame
    for widgets in frame.winfo_children():
```

```
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f = open('meter bridge.txt')
    l = f.readlines()
    f.close()

    title = Label(second_frame,text='Meter Bridge',font=('Times
New Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    aim = Label(second_frame,text=l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    aim.grid(row=1,column=0,sticky='w')

    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    matreqd.grid(row=2,column=0,sticky='w')

    formula =
Label(second_frame,text=l[4]+l[5]+l[6]+l[7]+l[8],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    formula.grid(row=3,column=0,sticky='w')

    procedure =
Label(second_frame,text=l[9]+l[10]+l[11]+l[12]+l[13]+l[14]+l[15]+l
[16],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
```

```
    procedure.grid(row=4,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The connections should be neat, clean and tight.
• Jockey should be gently slided over the bridge wire.
• Ensure balancing length are taken near 50cm to avoid end
resistance.
• Avoid length less than 35cm and greater than 65cm.
• Avoid continuous passage of current as the wire could get heated
up.
• While searching for balancig length, include H.R. to protect the
galvanometer.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    precautions.grid(row=5,column=0,sticky='w')

    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• End corrections are not considered.
• Meter bridge wire may not be of uniform cross section
area.\n''',font=('Times New Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=6,column=0,sticky='w')

    file = open('meterBridge.csv','w+',newline='')
    r = csv.reader(file)
    w = csv.writer(file)
    w.writerow(['Before/After Interchanging','R (ohm)','l
(cm)','100 - l (cm)','X (ohm)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
    obs.grid(row=7,column=0,sticky='w')

    before = Label(second_frame,text='1. Before
interchanging:',bg='white',font=('Times New
Roman',20),justify='left').grid(row=10,column=0,sticky='w')
```

```
    entry1 = Label(second_frame,text='R
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=11,column=0,sticky='w')
    entry2 = Label(second_frame,text='l
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=12,column=0,sticky='w')
    after = Label(second_frame,text='2. After
interchanging:',bg='white',font=('Times New
Roman',20),justify='left').grid(row=14,column=0,sticky='w')
    entry3 = Label(second_frame,text='R
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=15,column=0,sticky='w')
    entry4 = Label(second_frame,text='l
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=16,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row=11,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=12,column=0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input3.grid(row=15,column=0)
    input4 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input4.grid(row=16,column=0)

    def save1():
        tempvar1 = float(input1.get())
        tempvar2 = float(input2.get())
        l1=['before',tempvar1,tempvar2,100-
tempvar2,round(tempvar1*(round((100-tempvar2)/tempvar2,2)),3)]
        w.writerow(l1)
        messagebox.showinfo("Reading","Saved succesfully")
```

```
    def save2():
        tempvar3 = float(input3.get())
        tempvar4 = float(input4.get())
        l2=['after',tempvar3,tempvar4,100-
tempvar4,round(tempvar3*(round((100-tempvar4)/tempvar4,2)),3)]
        w.writerow(l2)
        messagebox.showinfo("Reading","Saved succesfully")
    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        x_before = []
        x_after = []
        for i in readings:
            if i[0] == 'before':
                x_before.append(float(i[4]))
            else:
                x_after.append(float(i[4]))
        mean1 = stat.mean(x_before)
        mean2 = stat.mean(x_after)
        label1 = Label(second_frame,text='Mean value of X before
interchanging = '+str(mean1)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=300,y=1310)
        label2 = Label(second_frame,text='Mean value of X after
interchanging = '+str(mean2)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=300,y=1485)

    Save1 = Button(second_frame,text='Save',width=15,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save1)
    Save1.grid(row=13,column=0,sticky='w')
    Save2 = Button(second_frame,text='Save',width=15,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save2)
    Save2.grid(row=17,column=0,sticky='w')
```

```
    res = Button(second_frame,text='Show
result',width=15,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    res.grid(row=18,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',
command=chemexpts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))

#Command for 'Meter Bridge - Series and Parallel' button
def meterBridgeSnP5():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')
```

```
    f = open('meter bridge snp.txt')
    l = f.readlines()
    f.close()

    title = Label(second_frame,text='Meter Bridge - Series and
Parallel',font=('Times New Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    aim = Label(second_frame,text=l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left')
    aim.grid(row=1,column=0,sticky='w')

    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left')
    matreqd.grid(row=2,column=0,sticky='w')

    formula =
Label(second_frame,text=l[4]+l[5]+l[6]+l[7]+l[8]+l[9]+l[10]+l[11]+
l[12],font=('Times New Roman',20),bg='white',justify='left')
    formula.grid(row=3,column=0,sticky='w')

    procedure =
Label(second_frame,text=l[13]+l[14]+l[15]+l[16]+l[17]+l[18],font=(
'Times New Roman',20),bg='white',justify='left')
    procedure.grid(row=4,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The connections should be neat, clean and tight.
• Jockey should be gently slided over the bridge wire.
• Ensure balancing length are taken near 50cm to avoid end
resistance.
• Avoid length less than 35cm and greater than 65cm.
• Avoid continuous passage of current as the wire could get heated
up.
```

```
• While searching for balancing length, include H.R. to protect
the galvanometer.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    precautions.grid(row=5,column=0,sticky='w')

    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• End corrections are not considered.
• Meter bridge wire may not be of uniform cross section
area.\n''',font=('Times New Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=6,column=0,sticky='w')

    file = open('meterBridgeSnP.csv','w+',newline='')
    r = csv.reader(file)
    w = csv.writer(file)
    w.writerow(['series/parallel/single','R (ohm)','l (cm)','100 -
l (cm)','X (ohm)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
    obs.grid(row=7,column=0,sticky='w')

    series = Label(second_frame,text='1. Single
cell:',bg='white',font=('Times New
Roman',20),justify='left').grid(row=8,column=0,sticky='w')
    entry1 = Label(second_frame,text='r₁
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=9,column=0,sticky='w')
    entry2 = Label(second_frame,text='r₂
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=10,column=0,sticky='w')
    parallel = Label(second_frame,text='2. Before interchanging r₁
and r₂ in series:',bg='white',font=('Times New
Roman',20),justify='left').grid(row=12,column=0,sticky='w')
```

```
    entry3 = Label(second_frame,text='R
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=13,column=0,sticky='w')
    entry4 = Label(second_frame,text='l
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=14,column=0,sticky='w')
    single = Label(second_frame,text='3. Before interchanging r1
and r2 in parallel:',bg='white',font=('Times New
Roman',20),justify='left').grid(row=16,column=0,sticky='w')
    entry5 = Label(second_frame,text='R
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=17,column=0,sticky='w')
    entry6 = Label(second_frame,text='l
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=18,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row=9,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=10,column=0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input3.grid(row=13,column=0)
    input4 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input4.grid(row=14,column=0)
    input5 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input5.grid(row=17,column=0)
    input6 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input6.grid(row=18,column=0)

    def save1():
```

```
        tempvar1 = float(input1.get())
        tempvar2 = float(input2.get())
        l1=['single',str(tempvar1)+';'+str(tempvar2),'-','-','-']
        w.writerow(l1)
        messagebox.showinfo("Reading","Saved succesfully")
    def save2():
        tempvar3 = float(input3.get())
        tempvar4 = float(input4.get())
        l1=['series',tempvar3,tempvar4,100-
tempvar4,round(tempvar3*(round((100-tempvar4)/tempvar4,2)),3)]
        w.writerow(l1)
        messagebox.showinfo("Reading","Saved succesfully")
    def save3():
        tempvar5 = float(input5.get())
        tempvar6 = float(input6.get())
        l1=['parallel',tempvar5,tempvar6,100-
tempvar6,round(tempvar5*(round((100-tempvar6)/tempvar6,2)),3)]
        w.writerow(l1)
        messagebox.showinfo("Reading","Saved succesfully")
    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        series = []
        parallel = []
        single = []
        for i in readings:
            if i[0] == 'series':
                series.append(float(i[4]))
            elif i[0] == 'parallel':
                parallel.append(float(i[4]))
            elif i[0] == 'single':
                l = i[1].split(';')
                single.extend(l)
        mean1 = stat.mean(series)
```

```
        mean2 = stat.mean(parallel)
        ser = float(single[0]) + float(single[1])
        par =
round((float(single[0])*float(single[1]))/(float(single[0]) +
float(single[1])),2)
        label1 = Label(second_frame,text='Mean value of X (r₁ and
r₂ in series) = '+str(mean1)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=600,y=1660)
        label2 = Label(second_frame,text='Mean value of X (r₁ and
r₂ in parallel) = '+str(mean2)+' \u2126',bg='white',font=('Times
New Roman',20)).place(x=600,y=1695)
        label3 = Label(second_frame,text='Effective resistance in
series = '+str(ser)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=600,y=1730)
        label4 = Label(second_frame,text='Effective resistance in
parallel = '+str(par)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=600,y=1765)

    Save1 = Button(second_frame,text='Save',width=15,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save1)
    Save1.grid(row=11,column=0,sticky='w')
    Save2 = Button(second_frame,text='Save',width=15,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save2)
    Save2.grid(row=15,column=0,sticky='w')
    Save3 = Button(second_frame,text='Save',width=15,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save3)
    Save3.grid(row=19,column=0,sticky='w')
    res = Button(second_frame,text='Show
result',width=15,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    res.grid(row=20,column=0,sticky='w')
```

```
    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))
```

#Command for 'Galvanometer' experiment

```
def galvano6():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f=open('galvanometer.txt')
    l=f.readlines()
    f.close()
```

```
    title = Label(second_frame,text='Galvanometer',font=('Times
New Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    aim = Label(second_frame,text=l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left')
    aim.grid(row=1,column=0,sticky='w')

    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left')
    matreqd.grid(row=2,column=0,sticky='w')

    formula =
Label(second_frame,text=l[4]+l[5]+l[6]+l[7]+l[8]+l[9]+l[10]+l[11]+
l[12]+l[13],font=('Times New Roman',20),bg='white',justify='left')
    formula.grid(row=3,column=0,sticky='w')

    procedure =
Label(second_frame,text=l[14]+l[15]+l[16]+l[17]+l[18]+l[19]+l[20]+
l[21]+l[22]+l[23] ,font=('Times New
Roman',20),bg='white',justify='left')
    procedure.grid(row=4,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The battery must have constant EMF.
• Before switching on the circuit, it should be ensured that the
value of R is high so that the current through the galvanometer is
not
   very high.
• All connection should be neat, clean and tight.
• Do not pass current throught the galvanometer for a long time as
it could be damaged.
• Ensure that the lower range resistance box is connected in
parallel and high range in series.
```

```
• While noting down the resistance for full deflection, make sure
only the series key is cose and parallel key is
open.\n''',font=('Times New Roman',20),bg='white',justify='left')
    precautions.grid(row=5,column=0,sticky='w')

    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• The resistance of the coils in the resistance box may not be
exactly the same as that marked.
• EMF of the battery may not be constant.
• Galvanometer divisions may not be of equal
size.\n''',font=('Times New Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=6,column=0,sticky='w')

    file = open('galvanometer.csv','w+',newline='')
    r = csv.reader(file)
    w = csv.writer(file)
    w.writerow(['theta (div)','R (ohm)','S (ohm)','G
(ohm)','(R+G)theta (ohm div)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
    obs.grid(row=7,column=0,sticky='w')

    emax = Label(second_frame,text='E
(V):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=8,column=0,sticky='w')
    maxdiv = Label(second_frame,text='\u03F4 max
(div):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=9,column=0,sticky='w')
    entry1 = Label(second_frame,text='\u03F4
(div):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=11,column=0,sticky='w')
```

```
    entry2 = Label(second_frame,text='R
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=12,column=0,sticky='w')
    entry3 = Label(second_frame,text='S
(\u2126):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=13,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row=8,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=9,column=0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input3.grid(row=11,column=0)
    input4 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input4.grid(row=12,column=0)
    input5 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input5.grid(row=13,column=0)

    def save1():
        messagebox.showinfo("Reading","Saved succesfully")
    def save2():
        tempvar3 = float(input3.get())
        tempvar4 = float(input4.get())
        tempvar5 = float(input5.get())
        rs = tempvar4*tempvar5
        r_s = tempvar4 - tempvar5

l1=[tempvar3,tempvar4,tempvar5,round(rs/r_s,2),round(tempvar3*(tem
pvar4+round(rs/r_s,2)),2)]
        w.writerow(l1)
        messagebox.showinfo("Reading","Saved succesfully")
```

```
    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        x = []
        y = []
        for i in readings:
            x.append(float(i[3]))
            y.append(float(i[4]))
        e = float(input1.get())
        mean1 = stat.mean(x)
        mean2 = stat.mean(y)
        k = round(e/mean2,8)
        label1 = Label(second_frame,text='Average G =
'+str(mean1)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=400,y=1720)
        label2 = Label(second_frame,text='Average (R+G)\u03F4 =
'+str(mean2)+' \u2126',bg='white',font=('Times New
Roman',20)).place(x=400,y=1755)
        label3 = Label(second_frame,text='Figure of merit (k) =
'+str(k),bg='white',font=('Times New
Roman',20)).place(x=400,y=1790)

    Save1 = Button(second_frame,text='Save',width=17,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save1)
    Save1.grid(row=10,column=0,sticky='w')
    Save2 = Button(second_frame,text='Save',width=17,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save2)
    Save2.grid(row=14,column=0,sticky='w')
    res = Button(second_frame,text='Show
result',width=17,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    res.grid(row=15,column=0,sticky='w')
```

```
    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))

#Command for 'Convex Mirror' experiment
def convexMirror7():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    f = open('convex mirror.txt')
    l = f.readlines()
    f.close()
```

```
    title = Label(second_frame,text='Convex Mirror',font=('Times
New Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    aim = Label(second_frame,text=l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    aim.grid(row=1,column=0,sticky='w')

    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left',wraplength=1500)
    matreqd.grid(row=2,column=0,sticky='w')

    formula =
Label(second_frame,text=l[4]+l[5]+l[6]+l[7]+l[8]+l[9]+l[10],font=(
'Times New Roman',20),bg='white',justify='left',wraplength=1500)
    formula.grid(row=3,column=0,sticky='w')

    procedure = Label(second_frame,text=l[11]+l[12],font=('Times
New Roman',20),bg='white',justify='left',wraplength=1500)
    procedure.grid(row=4,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The object, mirror, lens and the screen should be at the same
height.
• The principal axis of the mirror should be parallel to the
surface of the table.
• The horizontal and vertical cross wires should be equally clear
at image positions.
• The screen should be placed as close to the axis of the miror as
possible.
• The convex mirror should be placed close to the convex
lens.\n''',font=('Times New Roman',20),bg='white',justify='left')
    precautions.grid(row=5,column=0,sticky='w')
```

```
    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• Focal length of lens may not be small.
• Parallex eror while taking the readings.
• Spherical aberation of the mirror.
• Parallax error while taking reading
• Thickness of lens may cause some errors in the
observation.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=6,column=0,sticky='w')

    file = open('convexMirror.csv','w+',newline='')
    r = csv.reader(file)
    w = csv.writer(file)
    w.writerow(['Object Position (cm)','Lens Position
(cm)','Mirror Position (cm)','Screen Position (cm)','R (cm)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
    obs.grid(row=7,column=0,sticky='w')

    entry1 = Label(second_frame,text='Object position
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=8,column=0,sticky='w')
    entry2 = Label(second_frame,text='Lens position L
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=9,column=0,sticky='w')
    entry2 = Label(second_frame,text='Mirror position LM
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=10,column=0,sticky='w')
    entry3 = Label(second_frame,text='Screen position LS
(cm):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=11,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
```

```
    input1.grid(row=8,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=9,column=0)
    input3 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input3.grid(row=10,column=0)
    input4 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input4.grid(row=11,column=0)

    def save():
        tempvar1 = float(input1.get())
        tempvar2 = float(input2.get())
        tempvar3 = float(input3.get())
        tempvar4 = float(input4.get())
        l1=[tempvar1,tempvar2,tempvar3,tempvar4,tempvar4-tempvar3]
        w.writerow(l1)
        messagebox.showinfo("Reading","Saved succesfully")
    def result():
        file.seek(0)
        readings = list(r)
        readings.pop(0)
        R = []
        for i in readings:
            R.append(float(i[4]))
        mean = round(stat.mean(R),2)
        f = mean/2
        label1 = Label(second_frame,text='Mean value of R =
'+str(mean)+' cm',bg='white',font=('Times New
Roman',20)).place(x=500,y=1470)
        label2 = Label(second_frame,text='Mean value of f =
'+str(f)+' cm',bg='white',font=('Times New
Roman',20)).place(x=500,y=1520)
```

```
    Save = Button(second_frame,text='Save',width=26,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=12,column=0,sticky='w')
    res = Button(second_frame,text='Show
result',width=26,font=('Times New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=result)
    res.grid(row=14,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))
```

#Command for 'Diode Characteristics' experiment

```
def diode8():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()
    canvas = Canvas(frame)
    canvas.pack(side=LEFT,fill=BOTH,expand=1)
    scrollbar =
ttk.Scrollbar(frame,orient=VERTICAL,command=canvas.yview)
```

```
    scrollbar.pack(side=RIGHT,fill=Y)
    canvas.configure(yscrollcommand=scrollbar.set)
    second_frame = Frame(canvas,bg='white')
    canvas.create_window((0,0),window=second_frame,anchor='nw')

    title = Label(second_frame,text='Diode
Characteristics',font=('Times New Roman',25,'bold'),bg='white')
    title.grid(row=0,column=0,sticky='n')

    f=open('diode.txt')
    l=f.readlines()

    aim = Label(second_frame,text=l[0]+l[1],font=('Times New
Roman',20),bg='white',justify='left')
    aim.grid(row=1,column=0,sticky='w')

    matreqd = Label(second_frame,text=l[2]+l[3],font=('Times New
Roman',20),bg='white',justify='left')
    matreqd.grid(row=2,column=0,sticky='w')

    procedure = Label(second_frame,text=
l[4]+l[5]+l[6]+l[7]+l[8]+l[9]+l[10]+l[11]+l[12]+l[13]+l[14]+l[15],
font=('Times New Roman',20),bg='white',justify='left')
    procedure.grid(row=3,column=0,sticky='w')

    precautions = Label(second_frame,text='''PRECAUTIONS:
• The connections should be neat, clean and tight.
• The voltmeter and ammeter readings must be recorded accurately.
• Increase and decrease the voltage gradually.
• Do not pass current for a long time as it could heat up the
voltmeter.
• Always reduce voltage to zero before switching it off.
• Parallax error must be avoided.
• Key should be used in circuit and opened when the circuit is not
used.
```

```
• Reverse-biased voltage beyond breakdown should not be
applied.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    precautions.grid(row=4,column=0,sticky='w')

    sourcesoferror = Label(second_frame,text='''SOURCES OF ERROR:
• The diode may be faulty.
• The connections may be loose.\n''',font=('Times New
Roman',20),bg='white',justify='left')
    sourcesoferror.grid(row=5,column=0,sticky='w')

    f = open('diode.csv','w+',newline='')
    w = csv.writer(f)
    w.writerow(['Voltmeter (V)','Ammeter (mA)'])

    obs = Label(second_frame,text='''OBSERVATION:
Please enter your observations.''',font=('Times New
Roman',20),bg='white',justify='left')
    obs.grid(row=6,column=0,sticky='w')

    entry1 = Label(second_frame,text='Voltmeter Reading
(V):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=7,column=0,sticky='w')
    entry2 = Label(second_frame,text='Ammeter Reading
(mA):',bg='white',font=('Times New
Roman',20),justify='left').grid(row=8,column=0,sticky='w')
    input1 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input1.grid(row=7,column=0)
    input2 =
Entry(second_frame,bg='#ffe3f5',width=20,borderwidth=2)
    input2.grid(row=8,column=0)

    def save():
        tempvar1 = float(input1.get())
```

```
        tempvar2 = float(input2.get())
        l1=[tempvar1,tempvar2]
        w.writerow(l1)
        messagebox.showinfo("Reading","Saved succesfully")

    Save = Button(second_frame,text='Save',width=16,font=('Times
New
Roman',20),bg='#fdd6ff',activebackground='#ffffff',command=save)
    Save.grid(row=9,column=0,sticky='w')

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    p_home = Button(second_frame,text='P',width=3,font=('Times New
Roman',15),bg='#ff1694',activebackground='#ff70c5',command=phyexpt
s)
    p_home.place(x=2,y=2)
    c_home = Button(second_frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=52,y=2)

    root.update()
    canvas.configure(scrollregion=canvas.bbox('all'))

#For accessing list of physics experiments
def phyexpts():
    global root
    global frame
    for widgets in frame.winfo_children():
      widgets.destroy()

    label = Label(frame,text='LIST OF EXPERIMENTS',font=('Times
New Roman',25,'bold'),bg='white')
```

```
    label.place(x=580,y=10)

    expt1 = Button(frame,text='CONCAVE
MIRROR',bg='#ff0080',activebackground='#ff59ab',width=70,font=('Ti
mes New Roman',20,'bold'),command=concaveMirror1)
    expt1.place(x=200,y=60)

    expt2 = Button(frame,text='CONVEX
LENS',bg='#fec5e5',activebackground='#ff8fd9',width=70,font=('Time
s New Roman',20,'bold'),command=convexLens2)
    expt2.place(x=200,y=120)

    expt3 = Button(frame,text='OHM\'S
LAW',bg='#ff0080',activebackground='#ff59ab',width=70,font=('Times
New Roman',20,'bold'),command=ohmsLaw3)
    expt3.place(x=200,y=180)

    expt4 = Button(frame,text='METRE
BRIDGE',bg='#fec5e5',activebackground='#ff8fd9',width=70,font=('Ti
mes New Roman',20,'bold'),command=meterBridge4)
    expt4.place(x=200,y=240)

    expt5 = Button(frame,text='METRE BRIDGE - SERIES AND
PARALLEL',bg='#ff0080',activebackground='#ff59ab',width=70,font=('
Times New Roman',20,'bold'),command=meterBridgeSnP5)
    expt5.place(x=200,y=300)

    expt6 =
Button(frame,text='GALVANOMETER',bg='#fec5e5',activebackground='#f
f8fd9',width=70,font=('Times New
Roman',20,'bold'),command=galvano6)
    expt6.place(x=200,y=360)
```

```
    expt7 = Button(frame,text='CONVEX
MIRROR',bg='#ff0080',activebackground='#ff59ab',width=70,font=('Ti
mes New Roman',20,'bold'),command=convexMirror7)
    expt7.place(x=200,y=420)

    expt8 = Button(frame,text='DIODE
CHARACTERISTICS',bg='#fec5e5',activebackground='#ff8fd9',width=70,
font=('Times New Roman',20,'bold'),command=diode8)
    expt8.place(x=200,y=480)

    exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
    exitt.place(x=1471,y=2)
    c_home = Button(frame,text='C',width=3,font=('Times New
Roman',15),bg='#57fff5',activebackground='#96f4ff',command=chemexp
ts)
    c_home.place(x=2,y=2)

root = Tk()
root.geometry('1540x600')
root.title('Digital Lab Assistant')
root.configure(bg='white')

frame = Frame(root,bg='white')
frame.pack(fill=BOTH,expand=1)

label = Label(frame,text='DIGITAL LAB ASSISTANT',font=('Times New
Roman',25,'bold'),bg='white')
label.place(x=575,y=10)

phybutton =
Button(frame,text='PHYSICS',bg='#ff1694',activebackground='#ff70c5
',width=35,height=7,font=('Times New
Roman',25,'bold'),command=phyexpts)
```

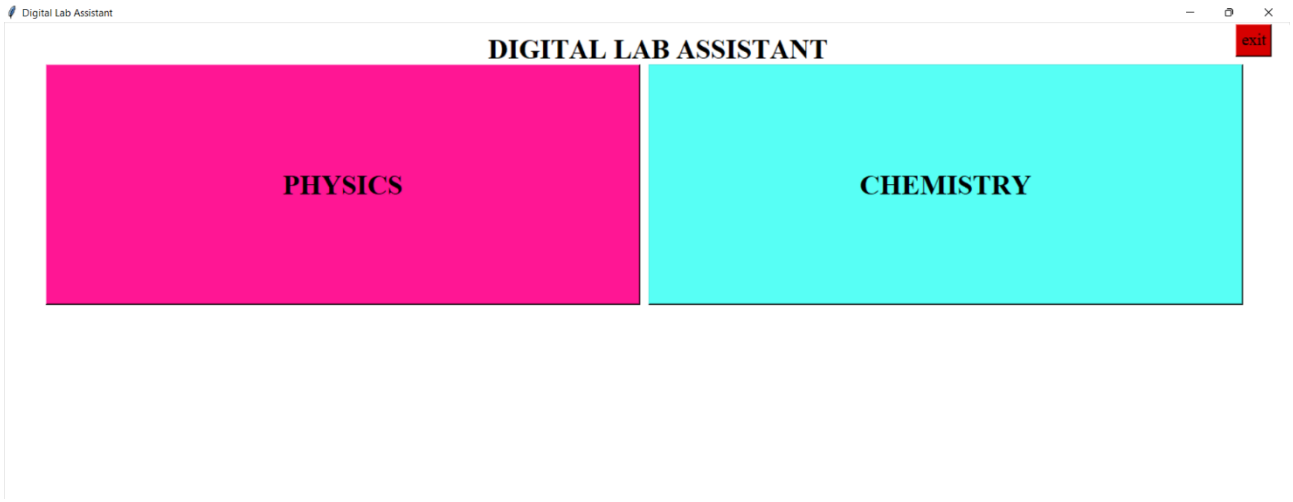
```
phybutton.place(x=50,y=50)

chembutton =
Button(frame,text='CHEMISTRY',bg='#57fff5',activebackground='#96f4
ff',width=35,height=7,font=('Times New
Roman',25,'bold'),command=chemexpts)
chembutton.place(x=770,y=50)

exitt = Button(frame,text='exit',width=3,font=('Times New
Roman',15),bg='#d60000',activebackground='#ffabab',command=root.de
stroy)
exitt.place(x=1471,y=2)
root.mainloop()
```

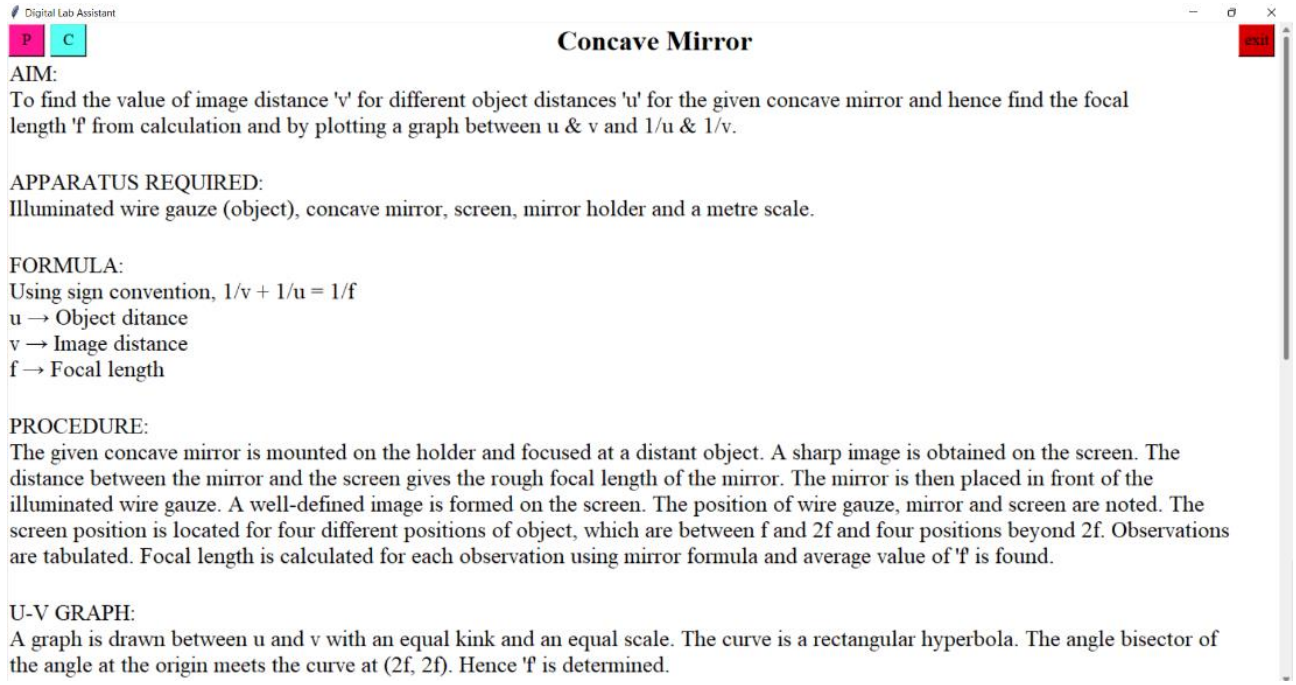
Sample Output

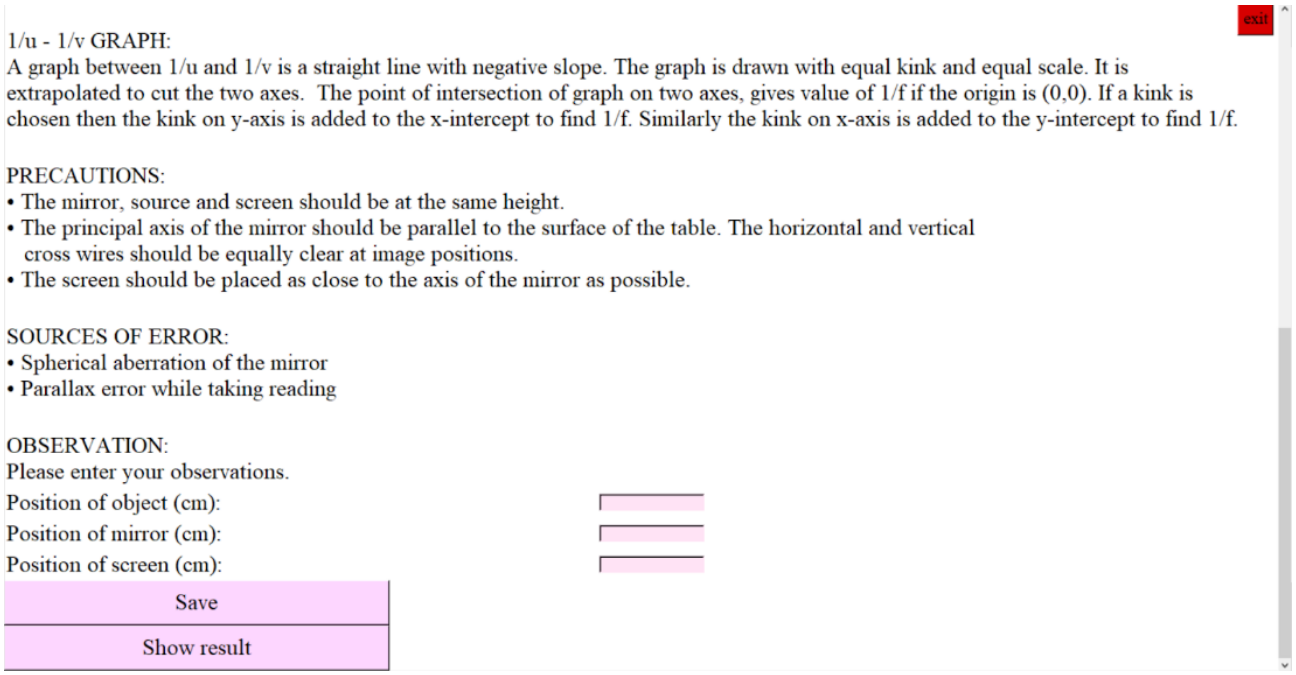
- Main Home Page



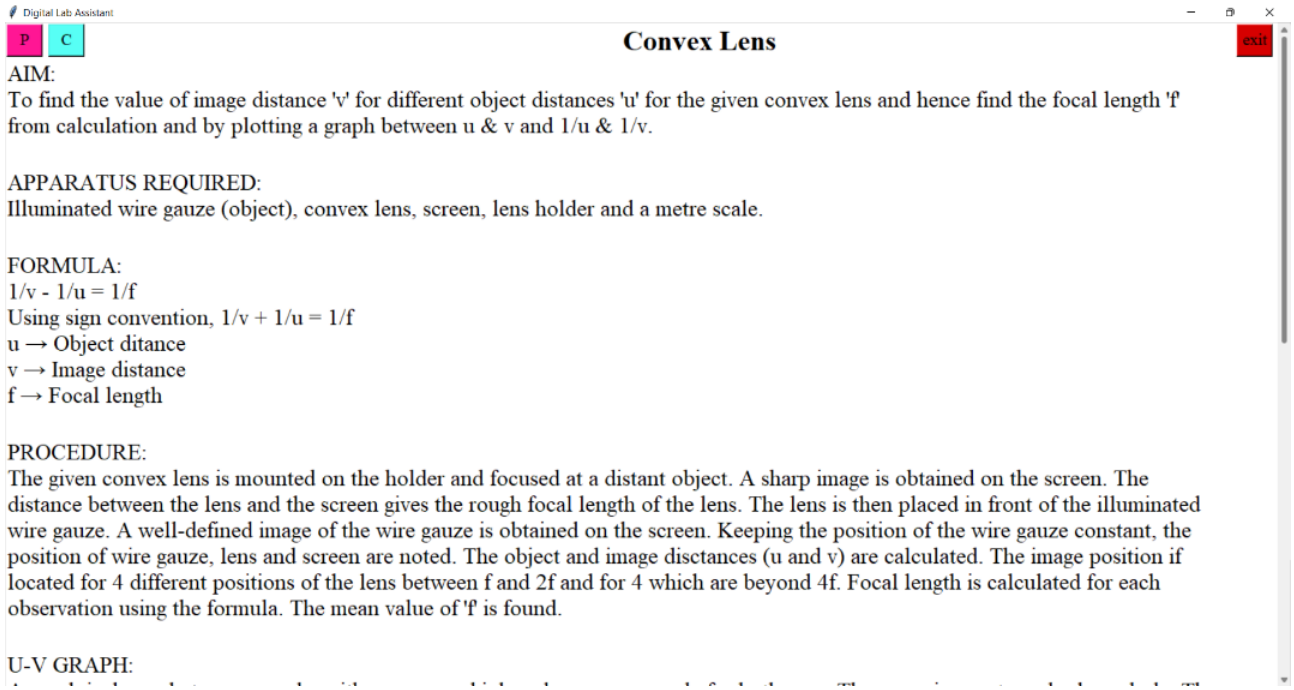
- Physics Home Page

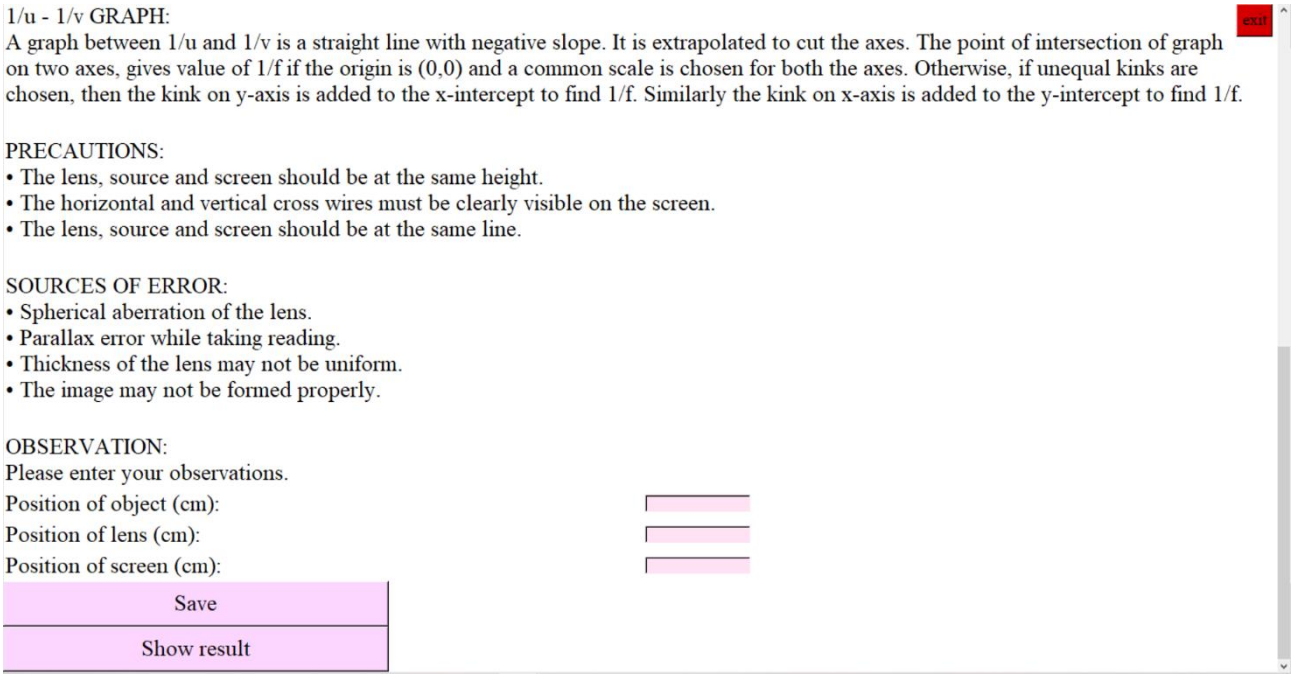
- Concave Mirror Experiment



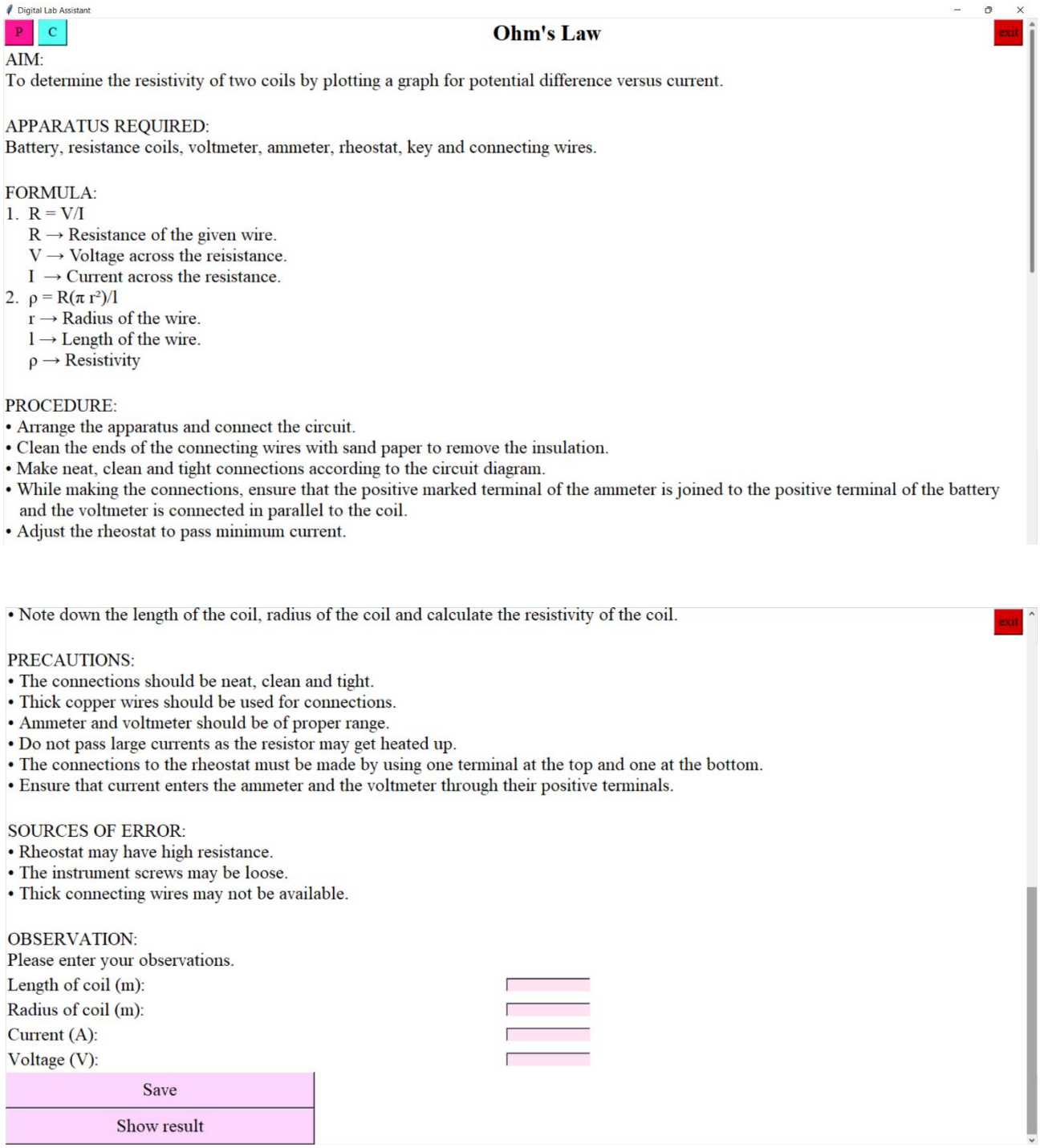


- Convex Lens

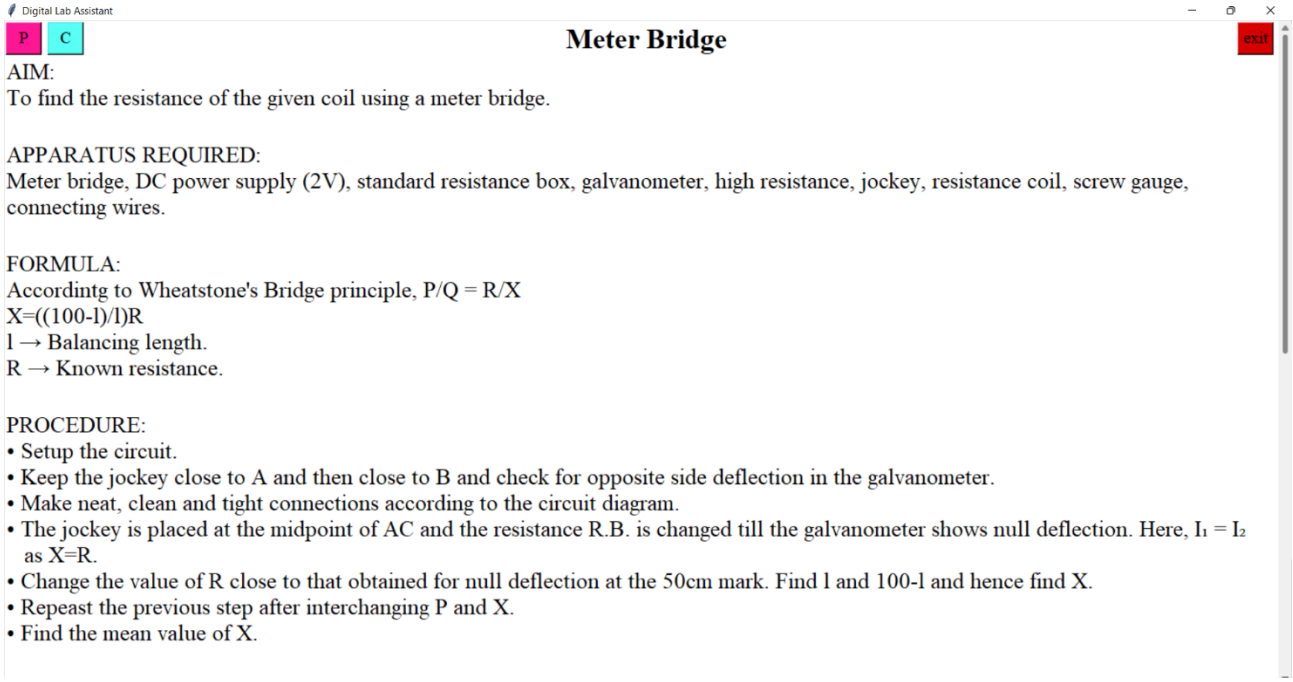

Digital Lab Assistant
P
C
Convex Lens
exit
AIM:
To find the value of image distance 'v' for different object distances 'u' for the given convex lens and hence find the focal length 'f' from calculation and by plotting a graph between u & v and 1/u & 1/v.
APPARATUS REQUIRED:
Illuminated wire gauze (object), convex lens, screen, lens holder and a metre scale.
FORMULA:
1/v - 1/u = 1/f
Using sign convention, 1/v + 1/u = 1/f
u → Object ditance
v → Image distance
f → Focal length
PROCEDURE:
The given convex lens is mounted on the holder and focused at a distant object. A sharp image is obtained on the screen. The distance between the lens and the screen gives the rough focal length of the lens. The lens is then placed in front of the illuminated wire gauze. A well-defined image of the wire gauze is obtained on the screen. Keeping the position of the wire gauze constant, the position of wire gauze, lens and screen are noted. The object and image disctances (u and v) are calculated. The image position if located for 4 different positions of the lens between f and 2f and for 4 which are beyond 4f. Focal length is calculated for each observation using the formula. The mean value of 'f' is found.
U-V GRAPH:

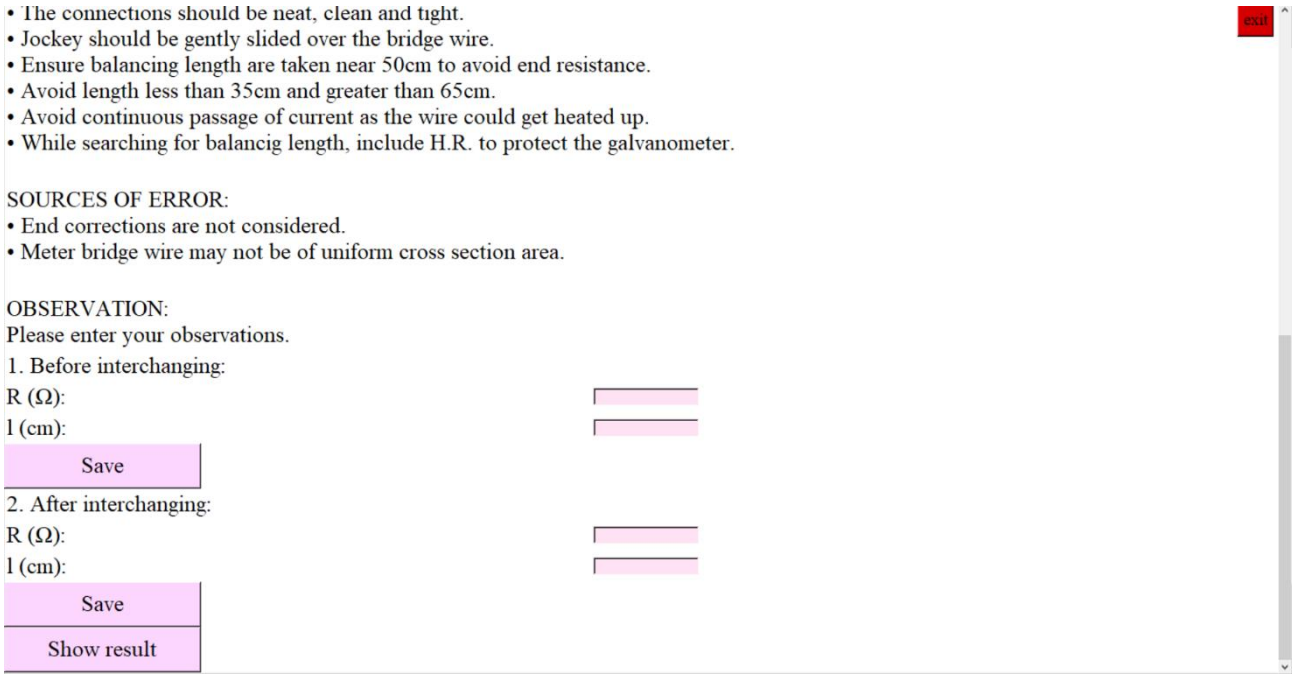

exit
1/u - 1/v GRAPH:
A graph between 1/u and 1/v is a straight line with negative slope. It is extrapolated to cut the axes. The point of intersection of graph on two axes, gives value of 1/f if the origin is (0,0) and a common scale is chosen for both the axes. Otherwise, if unequal kinks are chosen, then the kink on y-axis is added to the x-intercept to find 1/f. Similarly the kink on x-axis is added to the y-intercept to find 1/f.
PRECAUTIONS:
• The lens, source and screen should be at the same height.
• The horizontal and vertical cross wires must be clearly visible on the screen.
• The lens, source and screen should be at the same line.
SOURCES OF ERROR:
• Spherical aberration of the lens.
• Parallax error while taking reading.
• Thickness of the lens may not be uniform.
• The image may not be formed properly.
OBSERVATION:
Please enter your observations.
Position of object (cm):
Position of lens (cm):
Position of screen (cm):
Save
Show result

- Ohm's Law

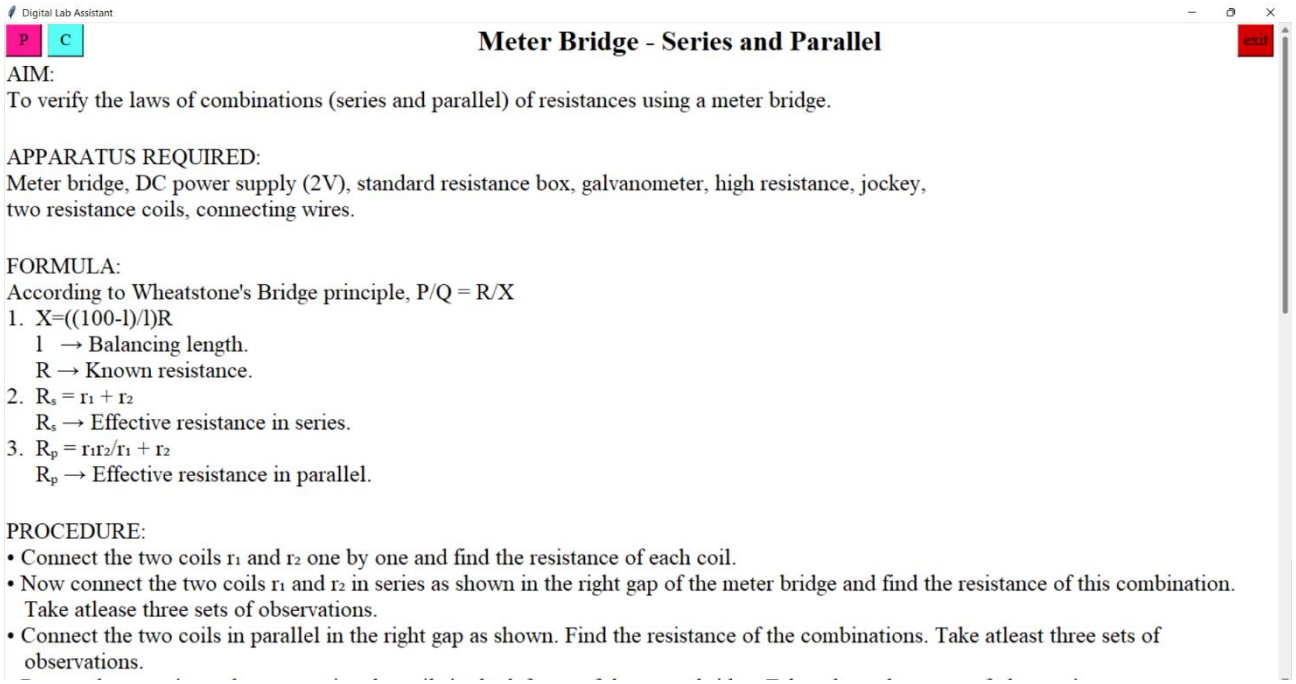

Digital Lab Assistant
P
C
Ohm's Law
exit
AIM:
To determine the resistivity of two coils by plotting a graph for potential difference versus current.
APPARATUS REQUIRED:
Battery, resistance coils, voltmeter, ammeter, rheostat, key and connecting wires.
FORMULA:
1. R = V/I
R → Resistance of the given wire.
V → Voltage across the reisistance.
I → Current across the resistance.
2. ρ = R(π r²)/l
r → Radius of the wire.
l → Length of the wire.
ρ → Resistivity
PROCEDURE:
• Arrange the apparatus and connect the circuit.
• Clean the ends of the connecting wires with sand paper to remove the insulation.
• Make neat, clean and tight connections according to the circuit diagram.
• While making the connections, ensure that the positive marked terminal of the ammeter is joined to the positive terminal of the battery and the voltmeter is connected in parallel to the coil.
• Adjust the rheostat to pass minimum current.
• Note down the length of the coil, radius of the coil and calculate the resistivity of the coil.
exit
PRECAUTIONS:
• The connections should be neat, clean and tight.
• Thick copper wires should be used for connections.
• Ammeter and voltmeter should be of proper range.
• Do not pass large currents as the resistor may get heated up.
• The connections to the rheostat must be made by using one terminal at the top and one at the bottom.
• Ensure that current enters the ammeter and the voltmeter through their positive terminals.
SOURCES OF ERROR:
• Rheostat may have high resistance.
• The instrument screws may be loose.
• Thick connecting wires may not be available.
OBSERVATION:
Please enter your observations.
Length of coil (m):
Radius of coil (m):
Current (A):
Voltage (V):
Save
Show result

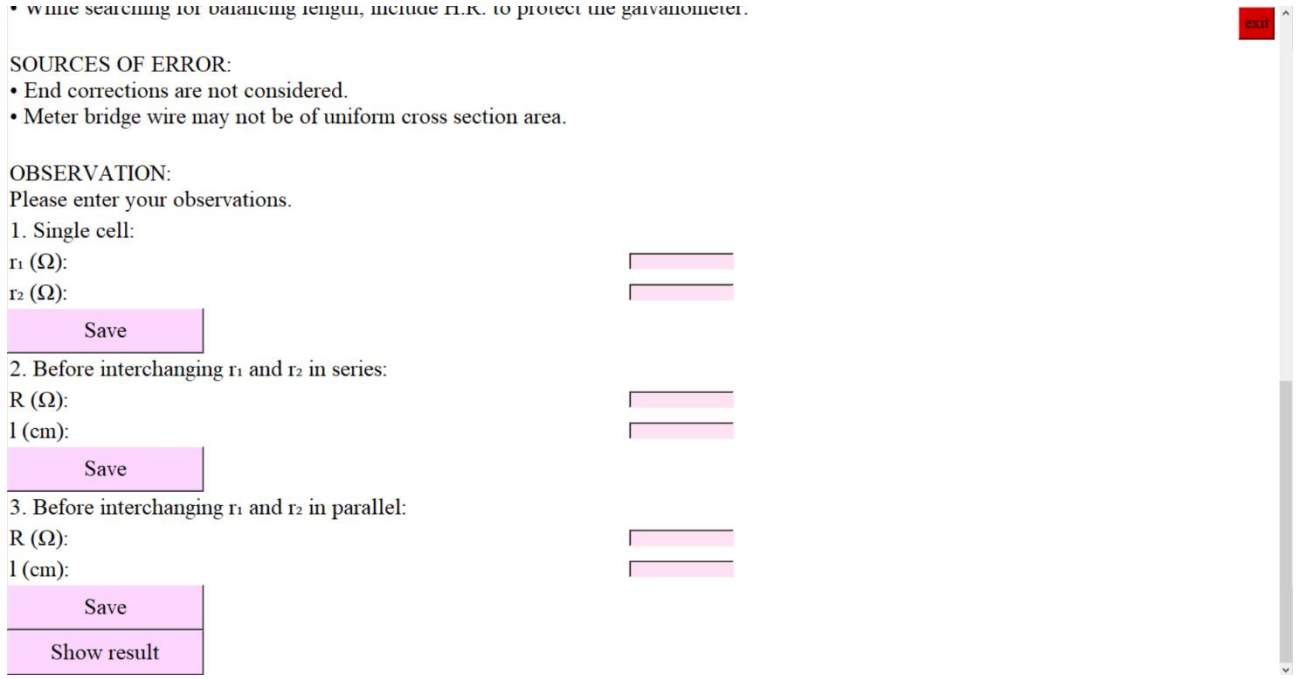
- Meter Bridge





- Meter Bridge - Series and Parallel





- Galvanometer

Digital Lab Assistant

P C

Galvanometer

exit

AIM:
To determine the resistance of a galvanometer by half deflection method and to find its figure of merit.

APPARATUS REQUIRED:
Galvanometer, DC power supply, high resistance box, variable low resistance box, keys.

FORMULA:
1. G = RS/R-S
   G → Resistance of galvanometer.
   R → High resistance.
   S → Low resistance.
2. Figure of merit (k) is defined as the amount of current required to produce one division of deflection in the scale of galvanometer.
   k = E/(R+G)Θ
   E → EMF of the cell
   Θ → deflection
   SI unit of k = Amp/div

PROCEDURE:
• The connections are made.
• The EMF of the source of electricity is noted.
• Plug out key K_2. This means S is not included in the circuit.
• Adjust the value of R so that there is almost full scale deflection of even number of divisions in the galvanometer. Note the deflection

exit

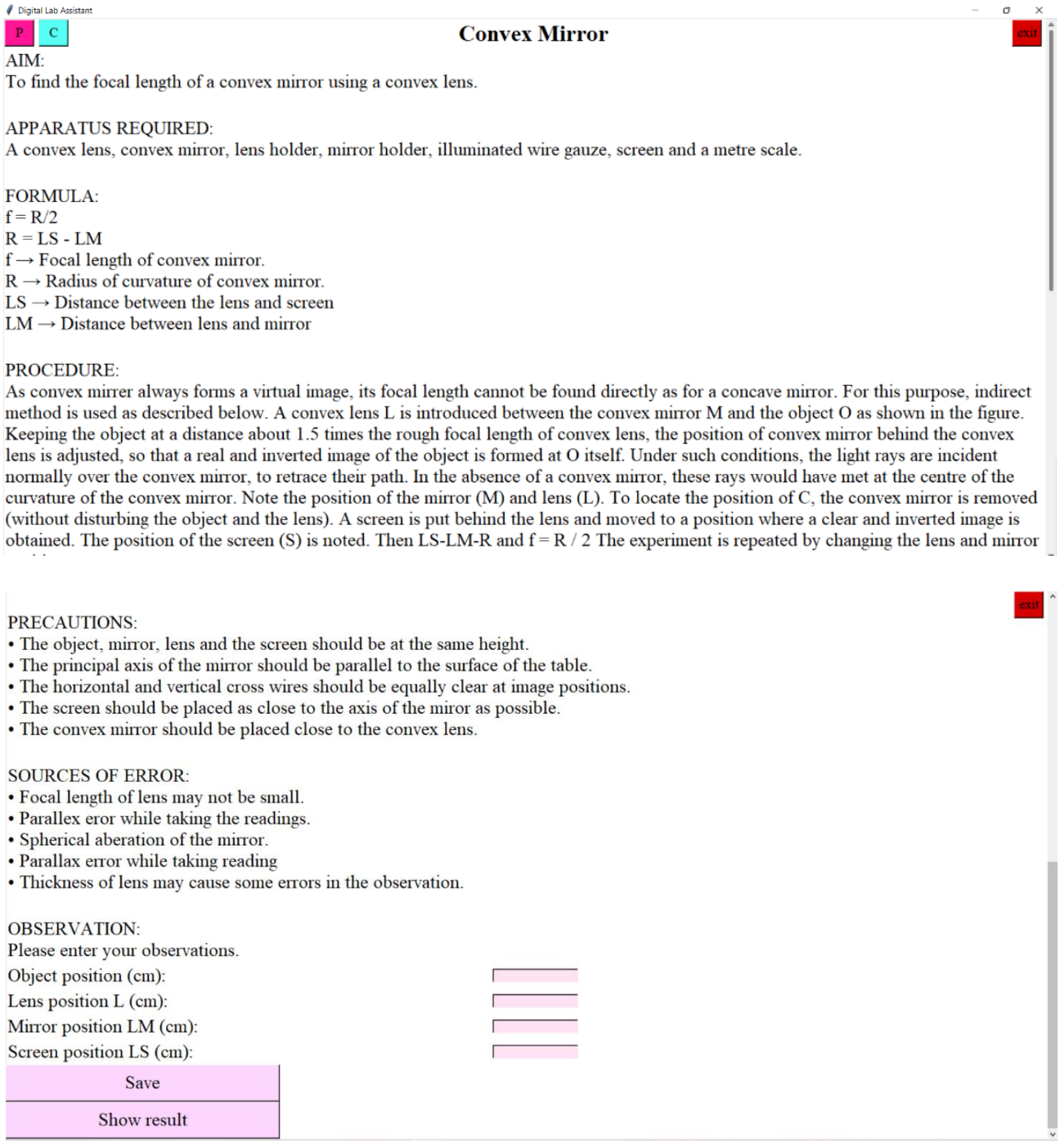
• Before switching on the circuit, it should be ensured that the value of R is high so that the current through the galvanometer is not very high.
• All connection should be neat, clean and tight.
• Do not pass current throught the galvanometer for a long time as it could be damaged.
• Ensure that the lower range resistance box is connected in parallel and high range in series.
• While noting down the resistance for full deflection, make sure only the series key is cose and parallel key is open.

SOURCES OF ERROR:
• The resistance of the coils in the resistance box may not be exactly the same as that marked.
• EMF of the battery may not be constant.
• Galvanometer divisions may not be of equal size.

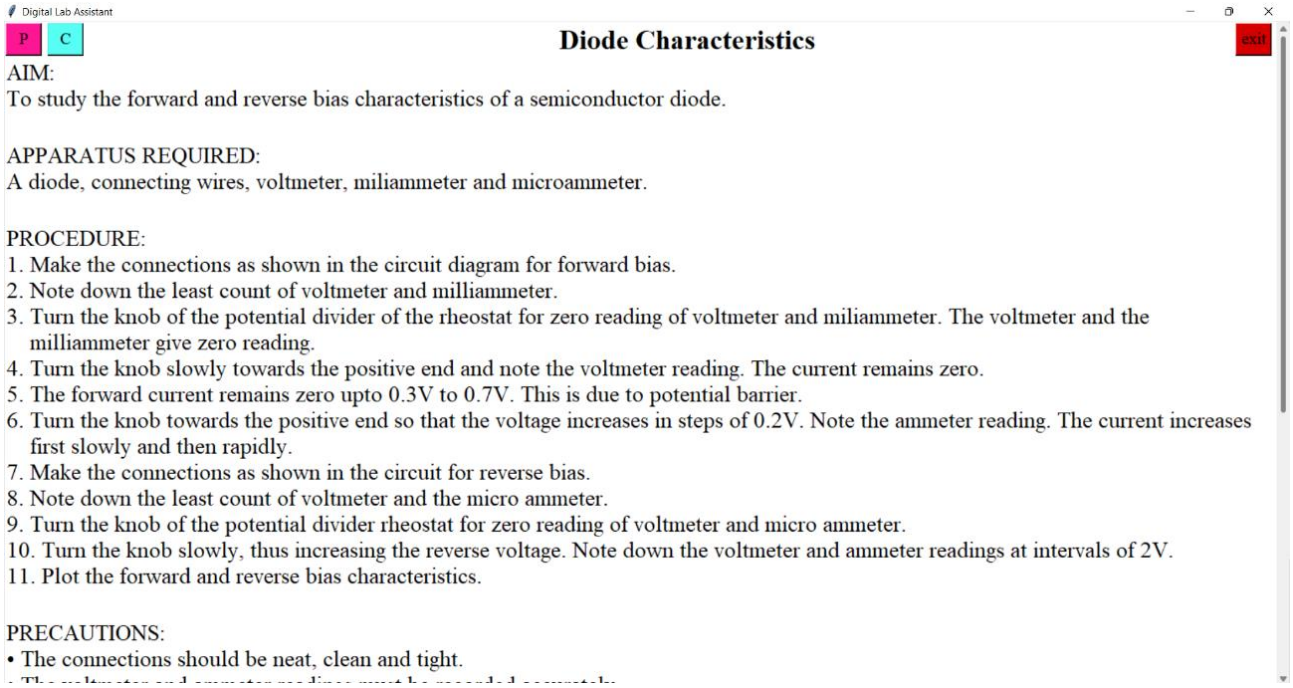
OBSERVATION:
Please enter your observations.

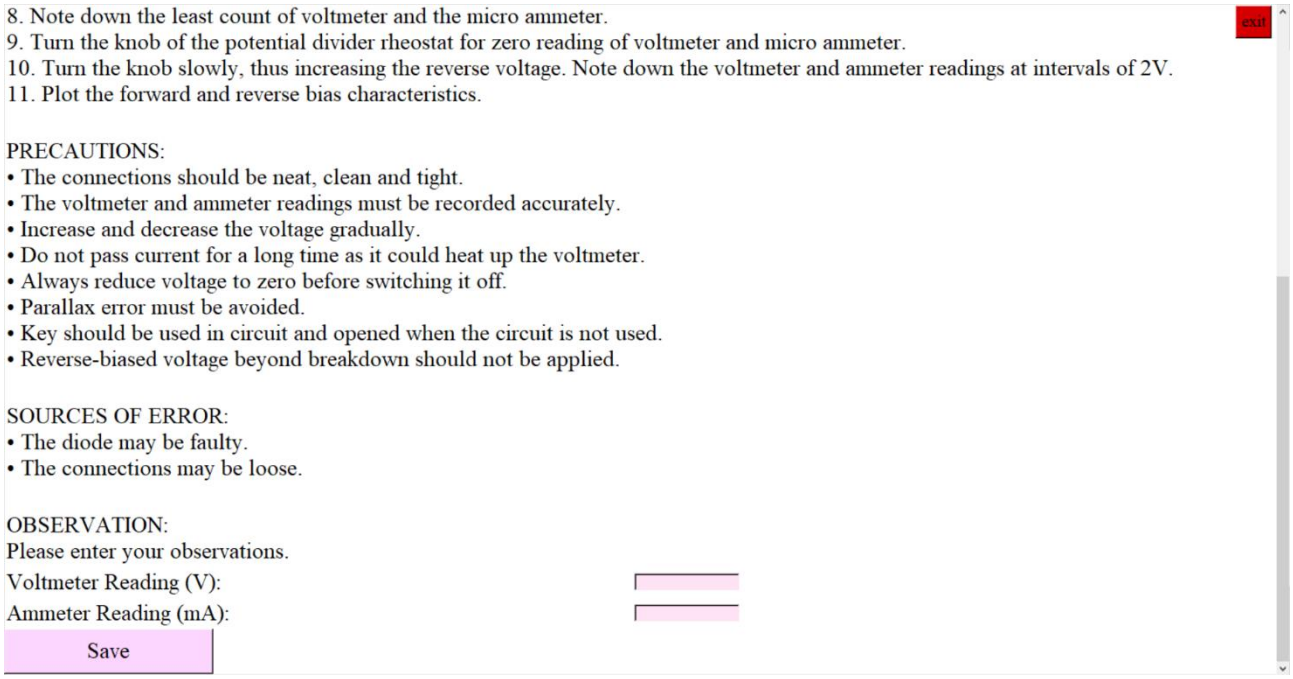
E (V):
Θ max (div):
Save
Θ (div):
R (Ω):
S (Ω):
Save
Show result

- Convex Mirror

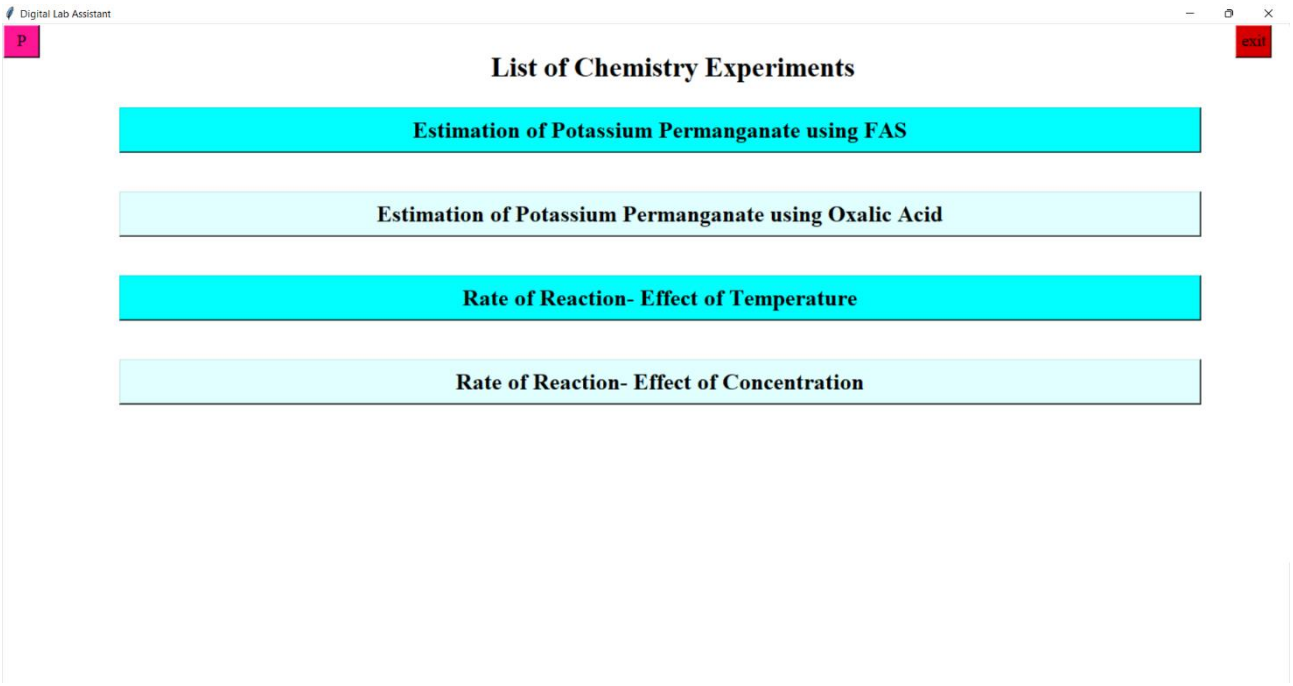


- Diode Characterestics

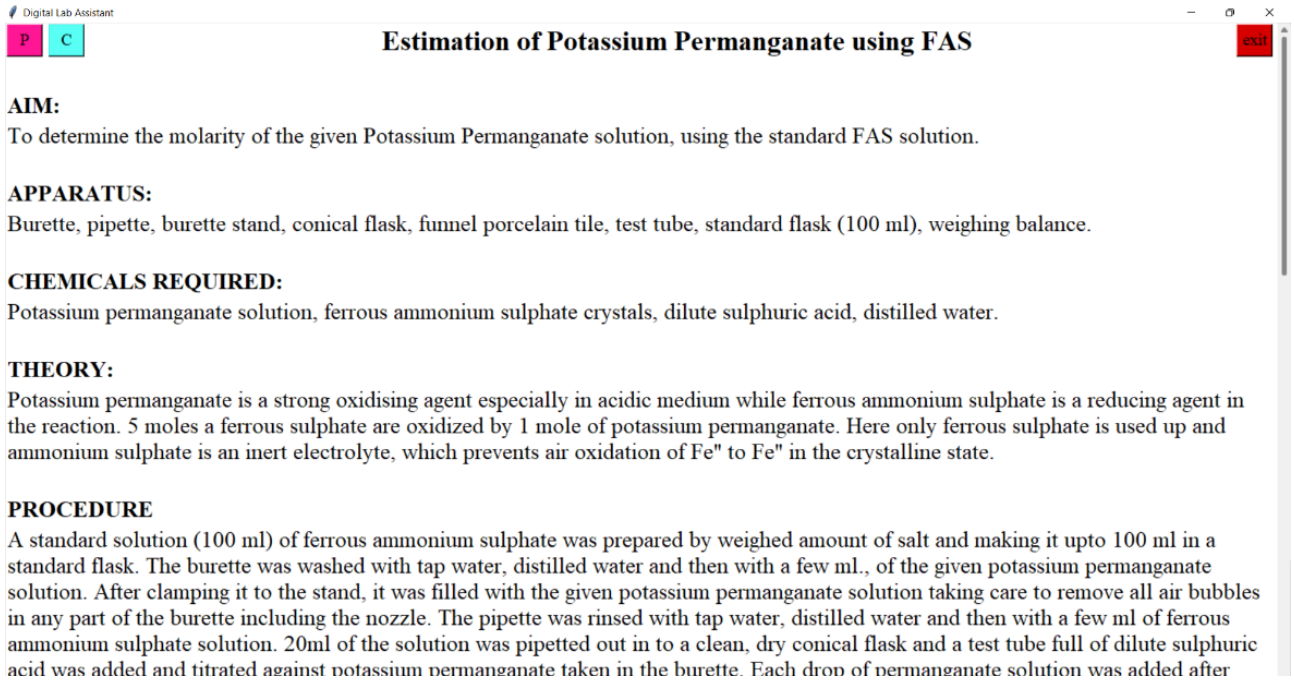




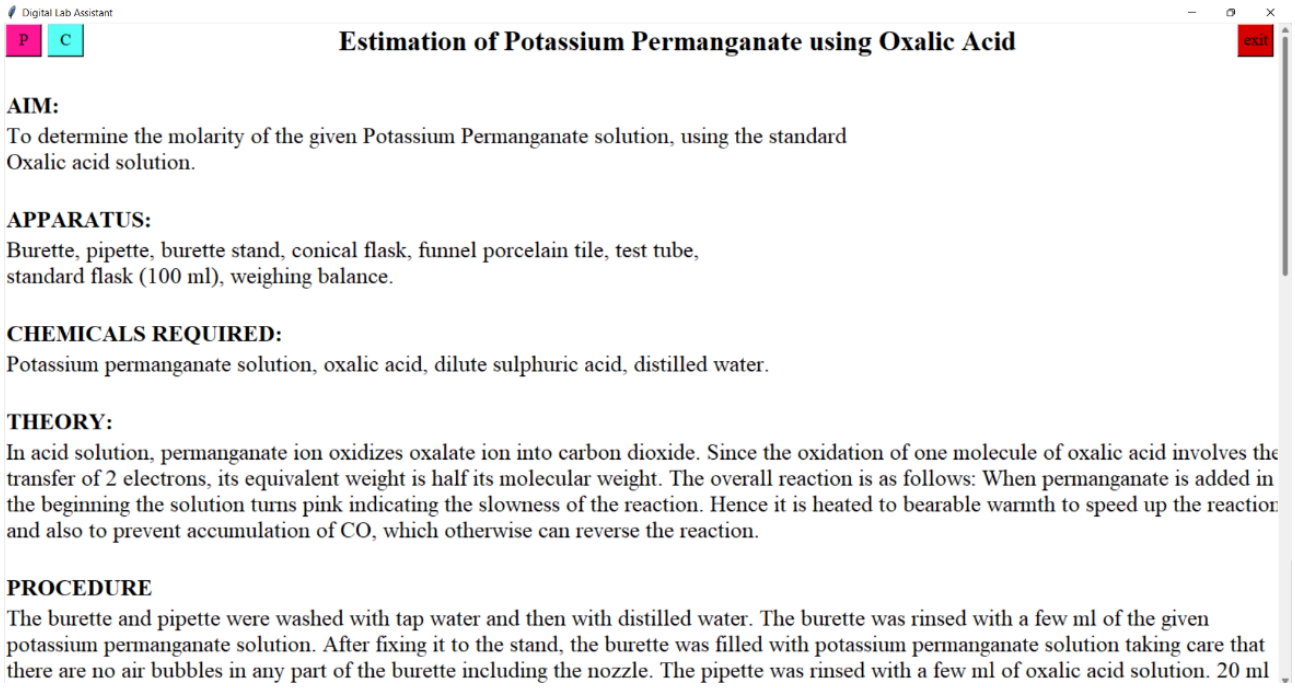
- Chemistry Home Page



- Estimation of Potassium Permanganate using FAS



- Estimation Of Potassium Permanganate using Oxalic Acid


Digital Lab Assistant

P C

Estimation of Potassium Permanganate using Oxalic Acid

exit

AIM:

To determine the molarity of the given Potassium Permanganate solution, using the standard Oxalic acid solution.

APPARATUS:

Burette, pipette, burette stand, conical flask, funnel porcelain tile, test tube, standard flask (100 ml), weighing balance.

CHEMICALS REQUIRED:

Potassium permanganate solution, oxalic acid, dilute sulphuric acid, distilled water.

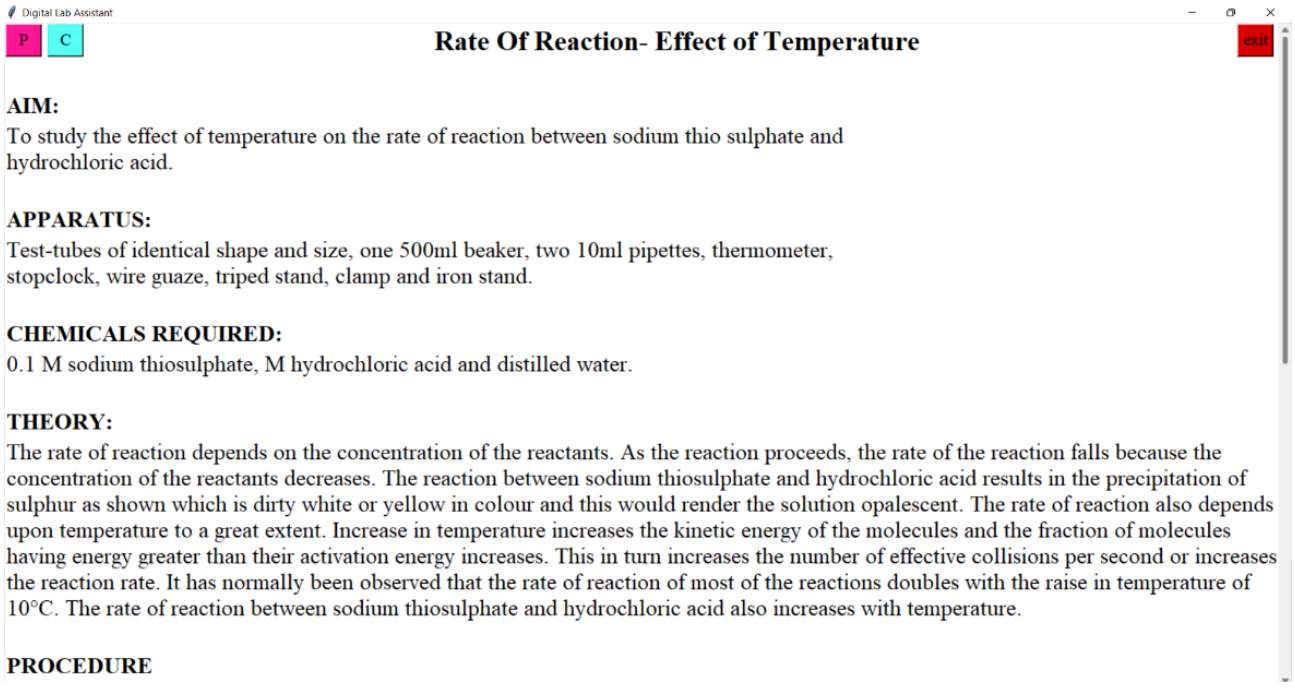
THEORY:

In acid solution, permanganate ion oxidizes oxalate ion into carbon dioxide. Since the oxidation of one molecule of oxalic acid involves the transfer of 2 electrons, its equivalent weight is half its molecular weight. The overall reaction is as follows: When permanganate is added in the beginning the solution turns pink indicating the slowness of the reaction. Hence it is heated to bearable warmth to speed up the reaction and also to prevent accumulation of CO, which otherwise can reverse the reaction.

PROCEDURE

The burette and pipette were washed with tap water and then with distilled water. The burette was rinsed with a few ml of the given potassium permanganate solution. After fixing it to the stand, the burette was filled with potassium permanganate solution taking care that there are no air bubbles in any part of the burette including the nozzle. The pipette was rinsed with a few ml of oxalic acid solution. 20 ml

- Rate of Reaction- Effect of Temperature


Digital Lab Assistant

P C

Rate Of Reaction- Effect of Temperature

exit

AIM:

To study the effect of temperature on the rate of reaction between sodium thio sulphate and hydrochloric acid.

APPARATUS:

Test-tubes of identical shape and size, one 500ml beaker, two 10ml pipettes, thermometer, stopclock, wire guaze, triped stand, clamp and iron stand.

CHEMICALS REQUIRED:

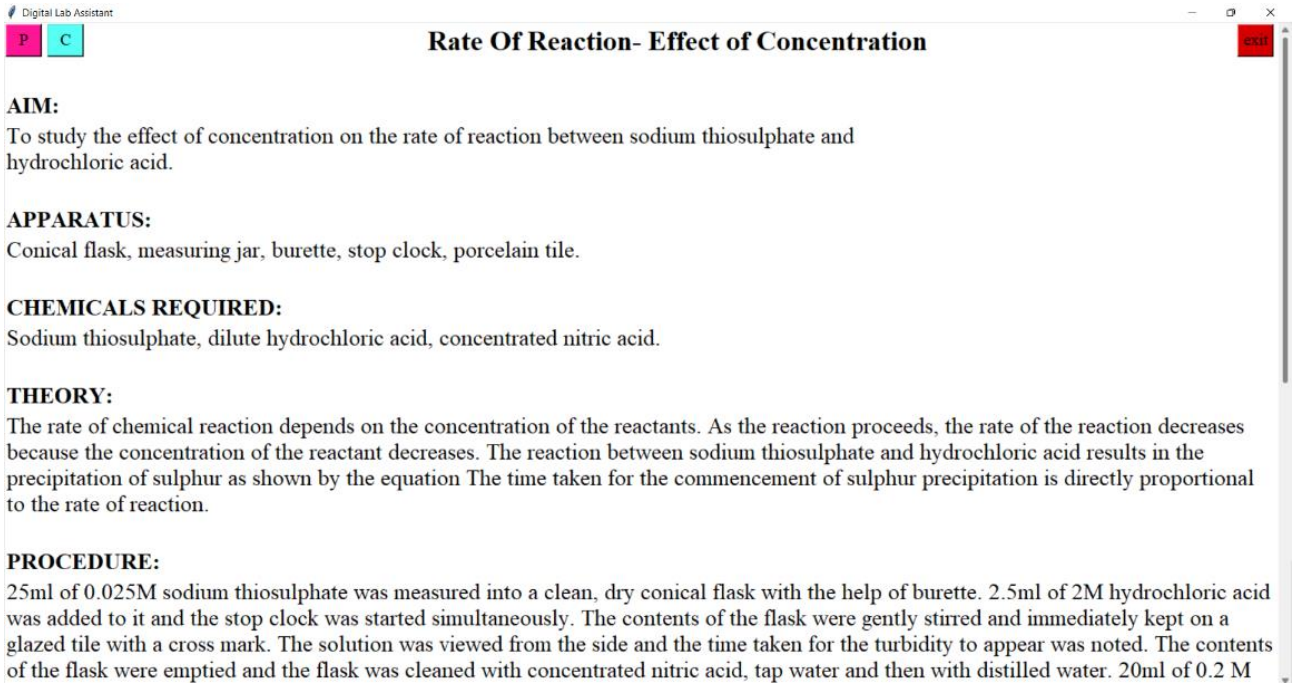
0.1 M sodium thiosulphate, M hydrochloric acid and distilled water.

THEORY:

The rate of reaction depends on the concentration of the reactants. As the reaction proceeds, the rate of the reaction falls because the concentration of the reactants decreases. The reaction between sodium thiosulphate and hydrochloric acid results in the precipitation of sulphur as shown which is dirty white or yellow in colour and this would render the solution opalescent. The rate of reaction also depends upon temperature to a great extent. Increase in temperature increases the kinetic energy of the molecules and the fraction of molecules having energy greater than their activation energy increases. This in turn increases the number of effective collisions per second or increases the reaction rate. It has normally been observed that the rate of reaction of most of the reactions doubles with the raise in temperature of 10°C. The rate of reaction between sodium thiosulphate and hydrochloric acid also increases with temperature.

PROCEDURE

- Rate of Reaction- Effect of Concentration


Digital Lab Assistant

P C

Rate Of Reaction- Effect of Concentration

exit

AIM:

To study the effect of concentration on the rate of reaction between sodium thiosulphate and hydrochloric acid.

APPARATUS:

Conical flask, measuring jar, burette, stop clock, porcelain tile.

CHEMICALS REQUIRED:

Sodium thiosulphate, dilute hydrochloric acid, concentrated nitric acid.

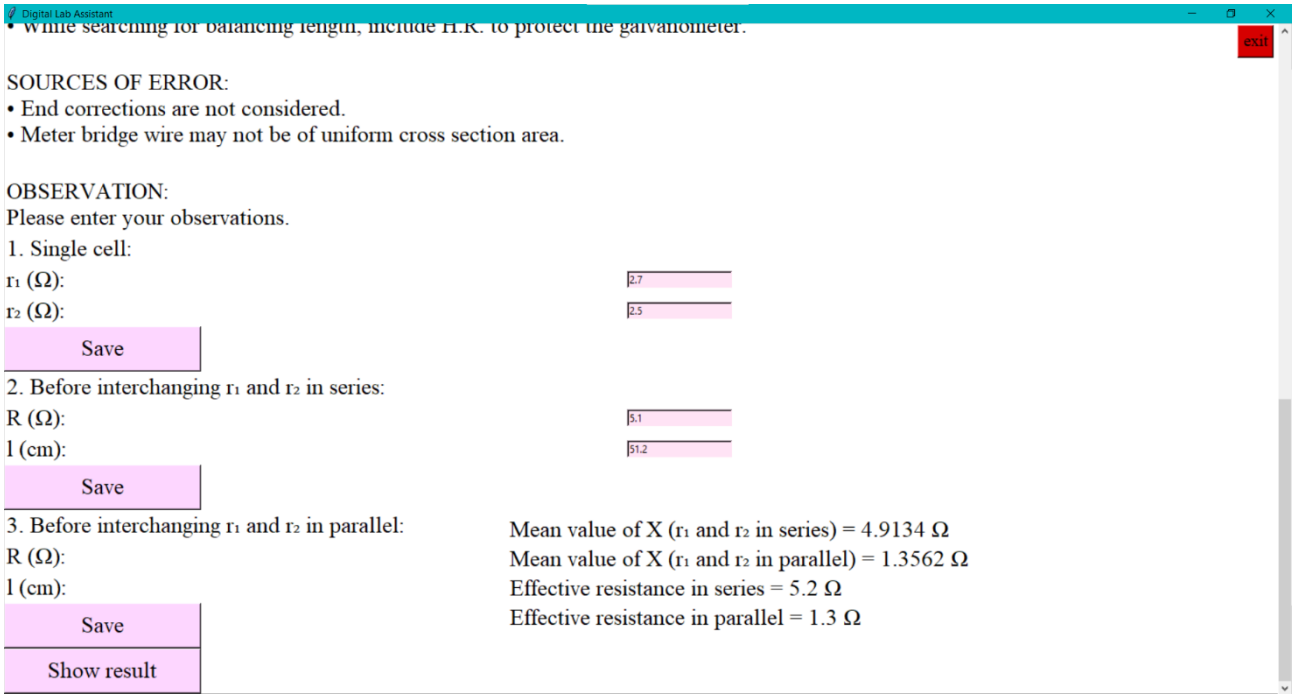
THEORY:

The rate of chemical reaction depends on the concentration of the reactants. As the reaction proceeds, the rate of the reaction decreases because the concentration of the reactant decreases. The reaction between sodium thiosulphate and hydrochloric acid results in the precipitation of sulphur as shown by the equation The time taken for the commencement of sulphur precipitation is directly proportional to the rate of reaction.

PROCEDURE:

25ml of 0.025M sodium thiosulphate was measured into a clean, dry conical flask with the help of burette. 2.5ml of 2M hydrochloric acid was added to it and the stop clock was started simultaneously. The contents of the flask were gently stirred and immediately kept on a glazed tile with a cross mark. The solution was viewed from the side and the time taken for the turbidity to appear was noted. The contents of the flask were emptied and the flask was cleaned with concentrated nitric acid, tap water and then with distilled water. 20ml of 0.2 M

- Sample Calculation:
  Physics


Digital Lab Assistant

• While searching for balancing length, include H.R. to protect the galvanometer.

exit

SOURCES OF ERROR:

• End corrections are not considered.
• Meter bridge wire may not be of uniform cross section area.

OBSERVATION:

Please enter your observations.

1. Single cell:

r_1 (Ω): 2.7

r_2 (Ω): 2.5

Save

2. Before interchanging r_1 and r_2 in series:

R (Ω): 5.1

l (cm): 51.2

Save

3. Before interchanging r_1 and r_2 in parallel:

R (Ω):

l (cm):

Save

Show result

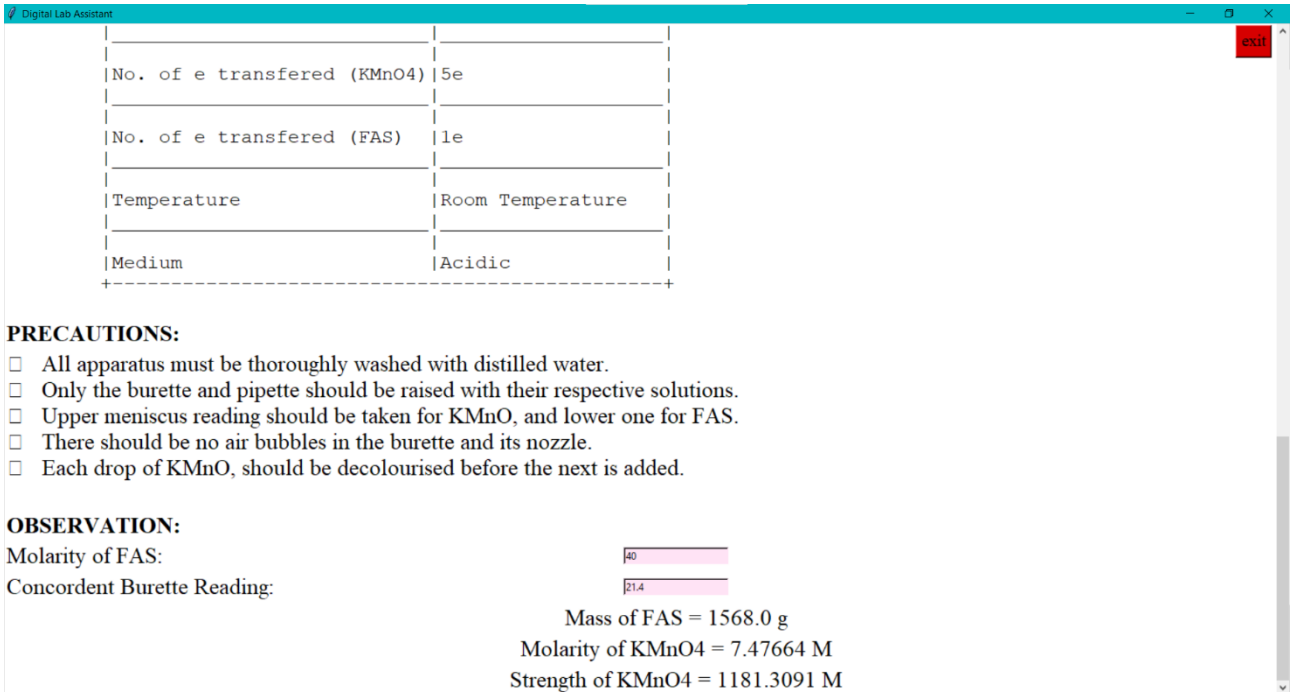
Mean value of X (r_1 and r_2 in series) = 4.9134 Ω

Mean value of X (r_1 and r_2 in parallel) = 1.3562 Ω

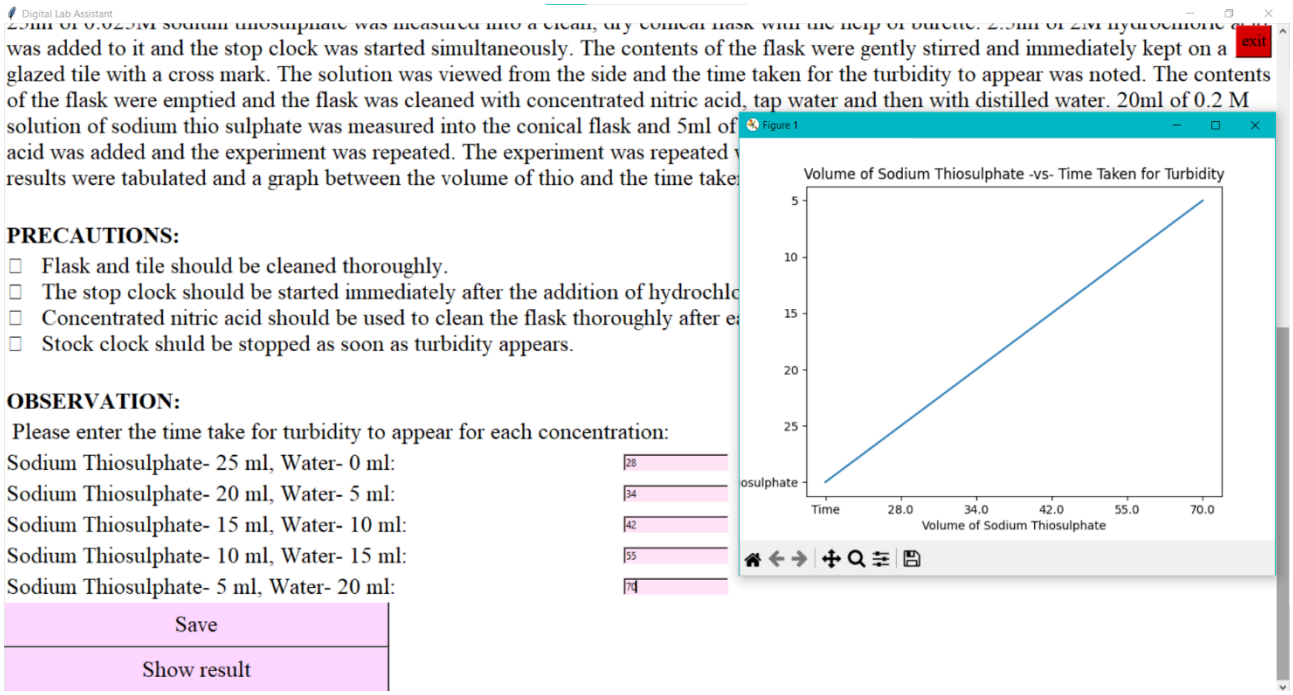
Effective resistance in series = 5.2 Ω

Effective resistance in parallel = 1.3 Ω

Chemistry



- Sample Graph



Challenges, Limitations and the Future

Challenges

The main challenges that we faced were

- Creating the 'P' and 'C' buttons, that would allow the user to switch between experiments without running the program again, every time.
- Implementing the scrollbar.
- Implementing file handling.

Limitations

- Pictures and more graphs could've been added while displaying the procedure for an experiment.
- The project only includes **some** experiments done in Class 11 and 12.
- The project could've been more efficient.

Future

This project can improved further in the future by

- Adding more experiments so that a larger number of students can use this application.
- Displaying pictures and more graphs in the experiments.
- Making the program more efficient, after learning more about Python.

Bibliography

- Computer Science with Python: Textbook for CBSE Class 12, Preeti Arora, Sultan Chand & Sons (P) Ltd, 2022.
- Computer Science With Python For Class 12: Sumita Arora, Dhanpat Rai & Co. (P) Ltd, 2022
- www.tutorialspoint.com
- www.pythonexamples.org
- www.geeksforgeeks.org
- www.pythontutorial.net
- www.stackoverflow.com
- www.askpython.com
- Codemy.com- Youtube
- www.activestate.com
- www.tutorialsteacher.com