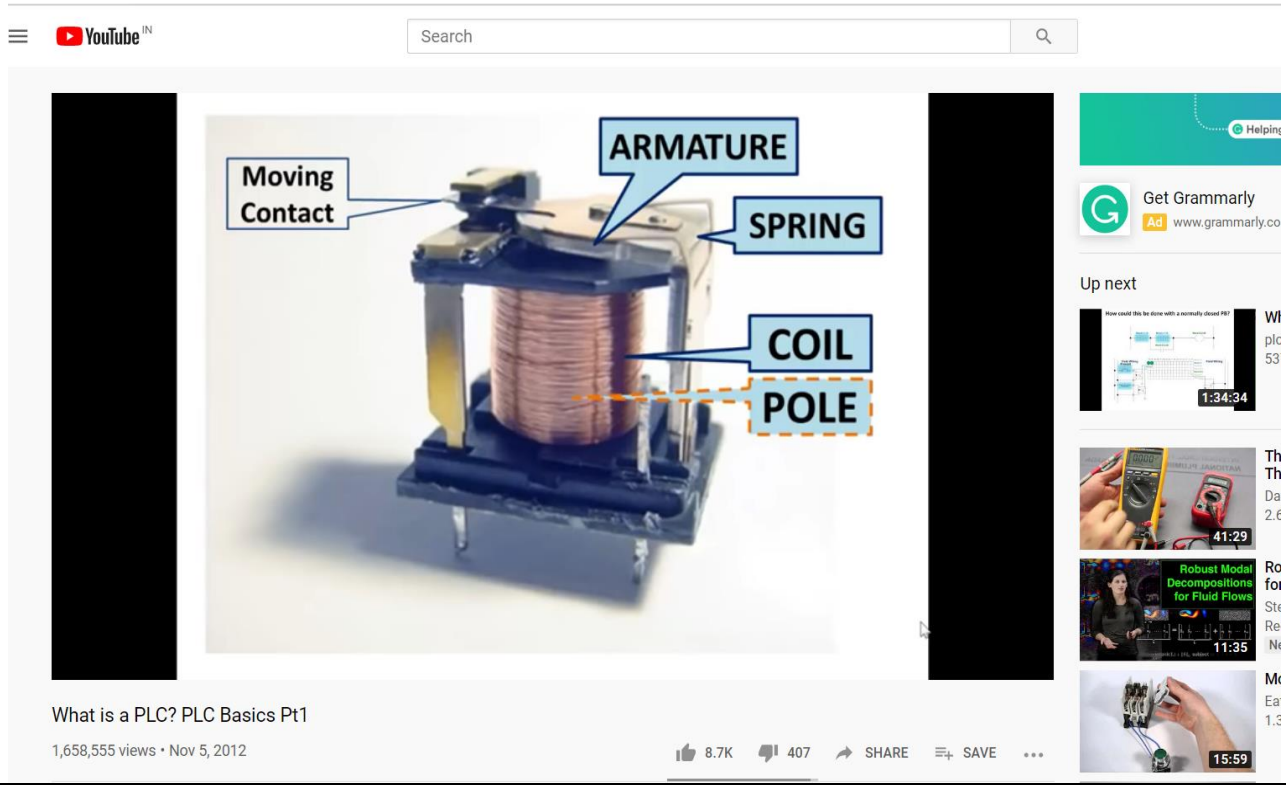
DAILY ASSESSMENT FORMAT

Date:	30-05-2020	Name:	BINDUSHRI
Course:	Logic designs	USN:	4AL17EC011
Topic:	1.applications of programmable logic controller.		6th A
Github Repository:	Bindushri		

FORENOON SESSION DETAILS



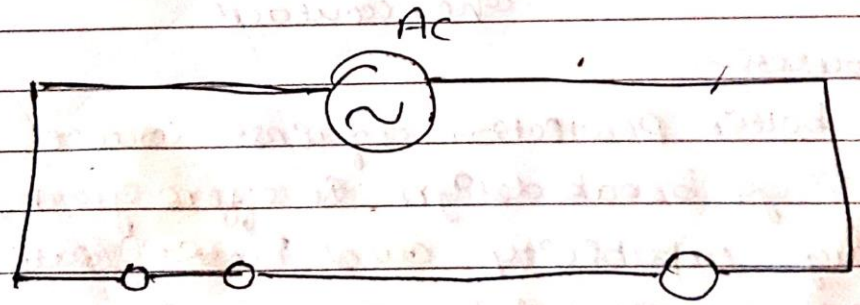
Day 3 - 30-05-2020.

Applications of programmable logic controller

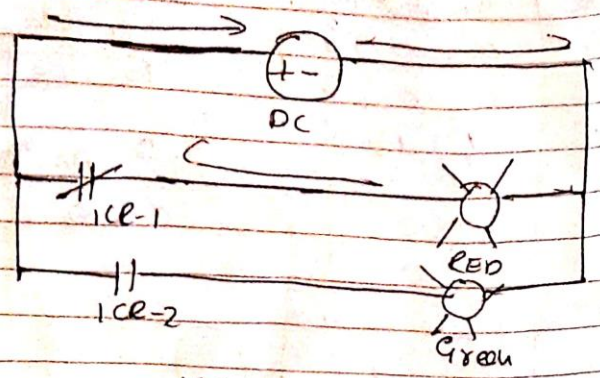
:

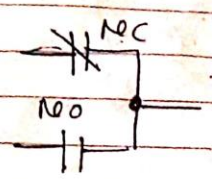
Operator Interface
- SPST switch
- Red Indicator
- Green Indicator

Power source
- Alternating current
- Direct current



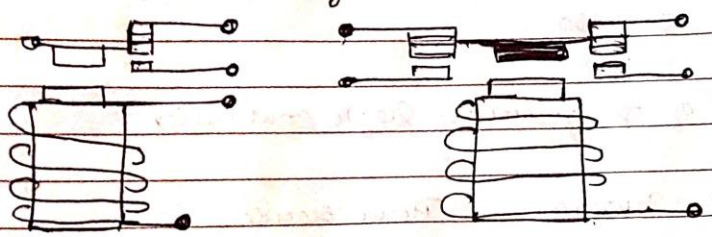
SPST switch. Contact Relay,





Normally closed contact — Normally open contact.

It is true for a small transition in the contacts are designed.



Rotates a common contact b/w the normally closed and the normally open — pulls a shorting bar b/w the normally closed and normally open contact

Double break contacts

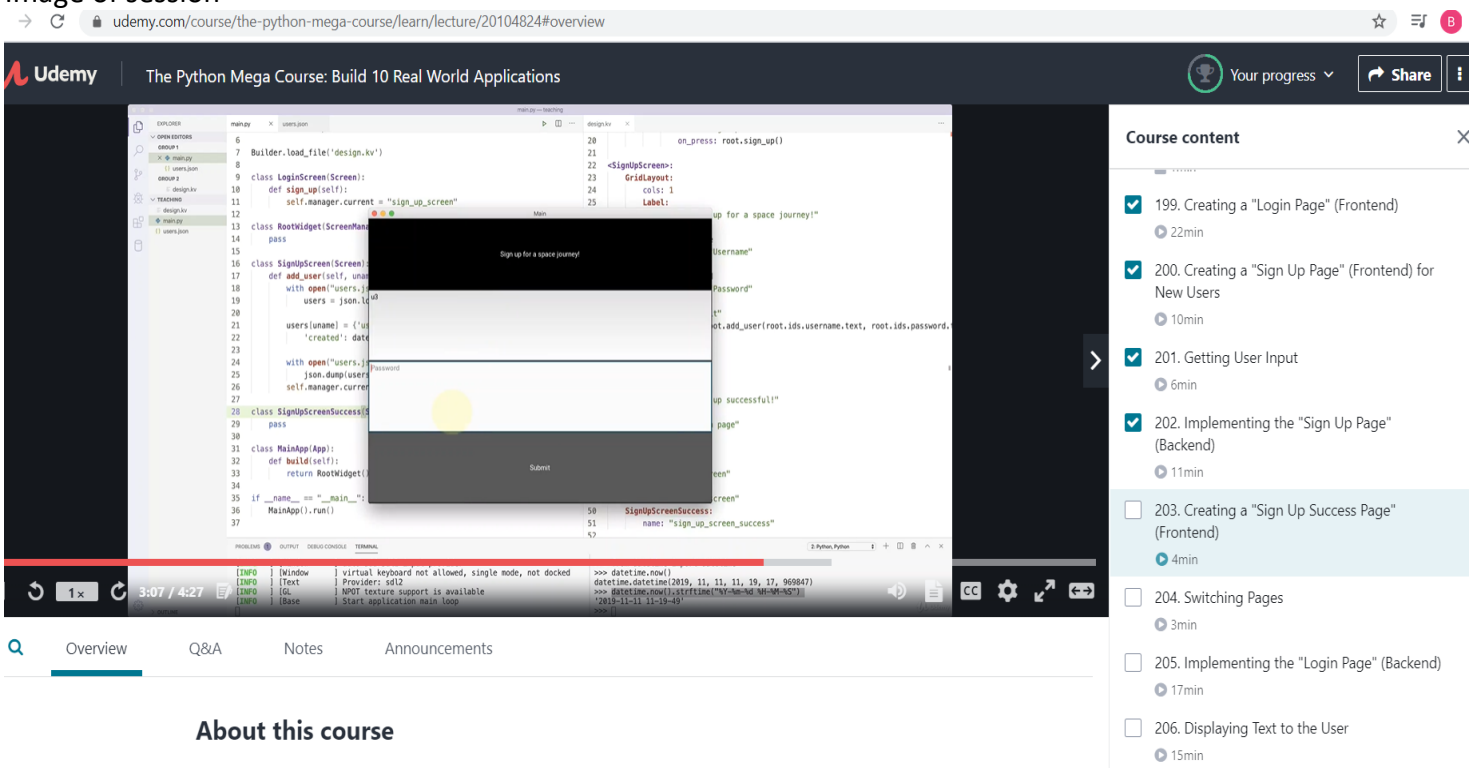
design provides better protection against contact welding than a single break design. It offers greater DC load breaking capability and better isolation this feature also provides separation of N.O and N.C circuits. Double-break contacts open the circuit in two places, creating two air gaps & reducing the probability of welded contacts by more than 50% compared to a single break design.

Date:30may2020
Course: python
Topic:
Basics:sec27

Name:Bindushri
USN:4AL17EC011
Sem&Sec:6th A

AFTERNOON SESSION DETAILS

Image of session



Date 20-05-2020.

Applications: Build a webcam motion Detector.

```
import cv2, time
from datetime import datetime

first_frame = None
status_list = [None, None]
time = []
df = pandas.DataFrame(columns = ["Start", "End"])
video = cv2.VideoCapture(0)

while True:
    check, frame = video.read()
    status = 0
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (21,21), 0)

    if first_frame is None:
        first_frame = gray
        continue

    delta_frame = cv2.absdiff(first_frame, gray)
    thresh_frame = cv2.threshold(delta_frame, 30, 255, cv2.
            THRESH_BINARY)[1]
    thresh_frame = cv2.dilate(thresh_frame, None,
            iterations = 2)

    (_, cnts, _) = cv2.findcontours(thresh_frame.copy(),
            cv2.RETR_EXTERNAL, cv2.CHAIN_
            APPROX_SIMPLE)
    for contour in cnts:
        if cv2.contourArea(contour) < 10000:
            continue
        status = 1
```

```
- (x, y, w, h) = cv2.boundingRect(contour)
- cv2.rectangle(frame, (x, y), (x+w, y+h),
                    (0, 255, 0), 3)
- status_list.append(status)

if status_list[-1] == 1 and status_list[-2] == 0:
    times.append(datetime.now())
if status_list[-1] == 0 and status_list[-2] == 1:
    times.append(datetime.now())

cv2.imshow("gray frame", gray)
cv2.imshow("Delta Frame", delta_frame)
cv2.imshow("Threshold Frame", thresh_frame)
cv2.imshow("Color Frame", frame)

key = cv2.waitKey(1)
if key == ord('q'):
    if status == 1:
        times.append(datetime.now())
        break
print(status_list)
print(times)
video.release()
cv2.destroyAllWindows
for i in range(0, len(times), 2):
    df = df.append({"Start": times[i], "End": times[i+1]},
                   ignore_index=True)

df.to_csv("Times.csv")

video.release()
cv2.destroyAllWindows
```

Excersise program:

acc.py - Untitled (Workspace) - Visual Studio Code

demo2 > account > acc.py

```python
class Account:


    def __init__(self,filepath):
     self.filepath=filepath
     with open(filepath,'r')as file:
        self.balance=int(file.read())


    def withdraw(self,ammount):
        self.balance=self.balance-ammount


    def deposit(self,ammount):
        self.balance=self.balance+ammount


    def commit(self):
        with open(self.filepath,'w')as fi
            file.write(str(self.balance))


account=Account("account//balance.txt")
print(account.balance)
account.deposit(100)
print(account.balance)
account.commit()
```

demo2 > account > balance.txt

```
1200
```

demo2 > backend.py > Database > __init__ > conn

```python
import sqlite3
class Database:

  def __init__(self,db):
    self.conn=sqlite3.connect(db)
    self.cur=self.conn.cursor()
    self.cur.execute("CREATE TABLE IF NOT EXISTS book (id I
    self.conn.commit()


  def insert(self,title,author,year,isbn):
    self.cur.execute("INSERT INTO book VALUES (NULL,?,?,?,?
    self.conn.commit()


  def view(self):
    self.cur.execute("SELECT * FROM book")
    rows=self.cur.fetchall()
    return rows

  def search(self,title="",author="",year="",isbn=""):
    self.cur.execute("SELECT * FROM book WHERE title=? OR a
    rows=cur.fetchall()
    return rows

  def delete(self,id):
    self.cur.execute("DELETE FROM book WHERE id=?",(id,))
    self.conn.commit()
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE SQL CONSOLE

```
    account.commit()
TypeError: commit() missing 1 required positional argument: 'path'
PS C:\Users\hp\Desktop\demo2> python account/acc.py
1000
1100
PS C:\Users\hp\Desktop\demo2> python account/acc.py
1100
1200
PS C:\Users\hp\Desktop\demo2>
```

acc.py - Untitled (Workspace) - Visual Studio Code

```python
class Account:


    def __init__(self,filepath):
     self.filepath=filepath
     with open(filepath,'r')as file:
        self.balance=int(file.read())


    def withdraw(self,ammount):
        self.balance=self.balance-ammount


    def deposit(self,ammount):
        self.balance=self.balance+ammount


    def commit(self):
        with open(self.filepath,'w')as fi
            file.write(str(self.balance))


account=Account("account//balance.txt")
print(account.balance)
account.withdraw(300)
print(account.balance)
account.commit()
```

```
900
```

```python
import sqlite3
class Database:

  def __init__(self,db):
    self.conn=sqlite3.connect(db)
    self.cur=self.conn.cursor()
    self.cur.execute("CREATE TABLE IF NOT EXISTS book (id I
    self.conn.commit()


  def insert(self,title,author,year,isbn):
    self.cur.execute("INSERT INTO book VALUES (NULL,?,?,?,?
    self.conn.commit()


  def view(self):
    self.cur.execute("SELECT * FROM book")
    rows=self.cur.fetchall()
    return rows

  def search(self,title="",author="",year="",isbn=""):
    self.cur.execute("SELECT * FROM book WHERE title=? OR a
    rows=cur.fetchall()
    return rows

  def delete(self,id):
    self.cur.execute("DELETE FROM book WHERE id=?",(id,))
    self.conn.commit()
```

```
1100
PS C:\Users\hp\Desktop\demo2> python account/acc.py
1100
1200
PS C:\Users\hp\Desktop\demo2> python account/acc.py
1200
900
PS C:\Users\hp\Desktop\demo2>
fwd-i-search: _
```

frontend.py

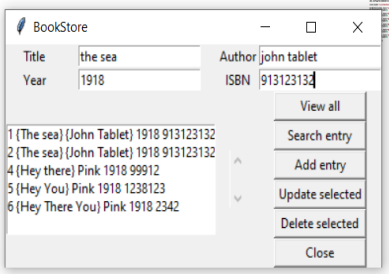
```python
from tkinter import *
import backend

def get_selected_row(event):
    global selected_tuple
    index=list1.curselection()[0]
    selected_tuple=list1.get(index)
    e1.delete(0,END)
    e1.insert(END,selected_tuple[1])
    e2.delete(0,END)
    e2.insert(END,selected_tuple[2])
    e3.delete(0,END)
    e3.insert(END,selected_tuple[3])
    e4.delete(0,END)
    e4.insert(END,selected_tuple[4])

def view_command():
    list1.delete(0,END)
    for row in backend.view():
        list1.insert(END,row)

def search_command():
    list1.delete(0,END)
    for row in backend.search(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()):
        list1.insert(END,row)

def add_command():
    backend.insert(title_text.get(),author_text.get(),year_text.get(),isbn_text.get())
    list1.delete(0,END)
    list1.insert(END,(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()))
```




```
  return self.func(*args)
File "frontend.py", line 33, in delete_command
  backend.delete(selected_tuple[0])
NameError: name 'selected_tuple' is not defined
PS C:\Users\hp\Desktop\demo2> python frontend.py
```

1.BANK ACCOUNT CODE:

```
class Account:


   def __init__(self,filepath):
    self.filepath=filepath
    with open(filepath,'r')as file:
       self.balance=int(file.read())


   def withdraw(self,ammount):
       self.balance=self.balance-ammount


   def deposit(self,ammount):
       self.balance=self.balance+ammount


   def commit(self):
       with open(self.filepath,'w')as file:
           file.write(str(self.balance))


account=Account("account//balance.txt")
print(account.balance)
account.withdraw(300)
print(account.balance)
account.commit()
```

1a)Backend.py:

```
import sqlite3
class Database:

  def __init__(self,db):
    self.conn=sqlite3.connect(db)
    self.cur=self.conn.cursor()
    self.cur.execute("CREATE TABLE IF NOT EXISTS book (id INTEGER PRIMARY KEY, title text,
author text, year integer, isbn integer)")
    self.conn.commit()
```

```
  def insert(self,title,author,year,isbn):
    self.cur.execute("INSERT INTO book VALUES (NULL,?,?,?,?)",(title,author,year,isbn))
    self.conn.commit()


  def view(self):
    self.cur.execute("SELECT * FROM book")
    rows=self.cur.fetchall()
    return rows

  def search(self,title="",author="",year="",isbn=""):
    self.cur.execute("SELECT * FROM book WHERE title=? OR author=? OR year=? OR isbn=?",
(title,author,year,isbn))
    rows=cur.fetchall()
    return rows

  def delete(self,id):
    self.cur.execute("DELETE FROM book WHERE id=?",(id,))
    self.conn.commit()

def update(self,id,title,author,year,isbn):
    self.cur.execute("UPDATE book SET title=?, author=?, year=?, isbn=? WHERE
id=?",(title,author,year,isbn,id))
    self.conn.commit()
```

1b)frontend.py:

```
from tkinter import *
from backend import Database

database=Database("books.db")

def get_selected_row(event):
    global selected_tuple
    index=list1.curselection()[0]
    selected_tuple=list1.get(index)
    e1.delete(0,END)
    e1.insert(END,selected_tuple[1])
    e2.delete(0,END)
    e2.insert(END,selected_tuple[2])
```

```
    e3.delete(0,END)
    e3.insert(END,selected_tuple[3])
    e4.delete(0,END)
    e4.insert(END,selected_tuple[4])

def view_command():
    list1.delete(0,END)
    for row in database.view():
        list1.insert(END,row)

def search_command():
    list1.delete(0,END)
    for row in database.search(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()):
        list1.insert(END,row)

def add_command():
    database.insert(title_text.get(),author_text.get(),year_text.get(),isbn_text.get())
    list1.delete(0,END)
    list1.insert(END,(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()))

def delete_command():
    database.delete(selected_tuple[0])

def update_command():
    database.update(selected_tuple[0],title_text.get(),author_text.get(),year_text.get(),isbn_text.get())

window=Tk()

window.wm_title("BookStore")

l1=Label(window,text="Title")
l1.grid(row=0,column=0)

l2=Label(window,text="Author")
l2.grid(row=0,column=2)

l3=Label(window,text="Year")
l3.grid(row=1,column=0)

l4=Label(window,text="ISBN")
l4.grid(row=1,column=2)

title_text=StringVar()
e1=Entry(window,textvariable=title_text)
e1.grid(row=0,column=1)
```

```
author_text=StringVar()
e2=Entry(window,textvariable=author_text)
e2.grid(row=0,column=3)

year_text=StringVar()
e3=Entry(window,textvariable=year_text)
e3.grid(row=1,column=1)

isbn_text=StringVar()
e4=Entry(window,textvariable=isbn_text)
e4.grid(row=1,column=3)

list1=Listbox(window, height=6,width=35)
list1.grid(row=2,column=0,rowspan=6,columnspan=2)

sb1=Scrollbar(window)
sb1.grid(row=2,column=2,rowspan=6)

list1.configure(yscrollcommand=sb1.set)
sb1.configure(command=list1.yview)

list1.bind('<<ListboxSelect>>',get_selected_row)

b1=Button(window,text="View all", width=12,command=view_command)
b1.grid(row=2,column=3)

b2=Button(window,text="Search entry", width=12,command=search_command)
b2.grid(row=3,column=3)

b3=Button(window,text="Add entry", width=12,command=add_command)
b3.grid(row=4,column=3)

b4=Button(window,text="Update selected", width=12,command=update_command)
b4.grid(row=5,column=3)

b5=Button(window,text="Delete selected", width=12,command=delete_command)
b5.grid(row=6,column=3)

b6=Button(window,text="Close", width=12,command=window.destroy)
b6.grid(row=7,column=3)

window.mainloop()
```

2A)TKINTER CODE:(frontend.py):

```
from tkinter import *
import backend

def get_selected_row(event):
   global selected_tuple
   index=list1.curselection()[0]
   selected_tuple=list1.get(index)
   e1.delete(0,END)
   e1.insert(END,selected_tuple[1])
   e2.delete(0,END)
   e2.insert(END,selected_tuple[2])
   e3.delete(0,END)
   e3.insert(END,selected_tuple[3])
   e4.delete(0,END)
   e4.insert(END,selected_tuple[4])

def view_command():
   list1.delete(0,END)
   for row in backend.view():
      list1.insert(END,row)

def search_command():
   list1.delete(0,END)
   for row in backend.search(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()):
      list1.insert(END,row)

def add_command():
   backend.insert(title_text.get(),author_text.get(),year_text.get(),isbn_text.get())
   list1.delete(0,END)
   list1.insert(END,(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()))

def delete_command():
   backend.delete(selected_tuple[0])

def update_command():
   backend.update(selected_tuple[0],title_text.get(),author_text.get(),year_text.get(),isbn_text.get())

window=Tk()

window.wm_title("BookStore")

l1=Label(window,text="Title")
l1.grid(row=0,column=0)
```

```
l2=Label(window,text="Author")
l2.grid(row=0,column=2)

l3=Label(window,text="Year")
l3.grid(row=1,column=0)

l4=Label(window,text="ISBN")
l4.grid(row=1,column=2)

title_text=StringVar()
e1=Entry(window,textvariable=title_text)
e1.grid(row=0,column=1)

author_text=StringVar()
e2=Entry(window,textvariable=author_text)
e2.grid(row=0,column=3)

year_text=StringVar()
e3=Entry(window,textvariable=year_text)
e3.grid(row=1,column=1)

isbn_text=StringVar()
e4=Entry(window,textvariable=isbn_text)
e4.grid(row=1,column=3)

list1=Listbox(window, height=6,width=35)
list1.grid(row=2,column=0,rowspan=6,columnspan=2)

sb1=Scrollbar(window)
sb1.grid(row=2,column=2,rowspan=6)

list1.configure(yscrollcommand=sb1.set)
sb1.configure(command=list1.yview)

list1.bind('<<ListboxSelect>>',get_selected_row)

b1=Button(window,text="View all", width=12,command=view_command)
b1.grid(row=2,column=3)

b2=Button(window,text="Search entry", width=12,command=search_command)
b2.grid(row=3,column=3)

b3=Button(window,text="Add entry", width=12,command=add_command)
b3.grid(row=4,column=3)

b4=Button(window,text="Update selected", width=12,command=update_command)
```

```
b4.grid(row=5,column=3)

b5=Button(window,text="Delete selected", width=12,command=delete_command)
b5.grid(row=6,column=3)

b6=Button(window,text="Close", width=12,command=window.destroy)
b6.grid(row=7,column=3)

window.mainloop()
```

2B)backend.py:

```
import sqlite3

def connect():
   conn=sqlite3.connect("books.db")
   cur=conn.cursor()
   cur.execute("CREATE TABLE IF NOT EXISTS book (id INTEGER PRIMARY KEY, title text, author text, year integer,
isbn integer)")
   conn.commit()
   conn.close()

def insert(title,author,year,isbn):
   conn=sqlite3.connect("books.db")
   cur=conn.cursor()
   cur.execute("INSERT INTO book VALUES (NULL,?,?,?,?)",(title,author,year,isbn))
   conn.commit()
   conn.close()
   view()

def view():
   conn=sqlite3.connect("books.db")
   cur=conn.cursor()
   cur.execute("SELECT * FROM book")
   rows=cur.fetchall()
   conn.close()
   return rows

def search(title="",author="",year="",isbn=""):
   conn=sqlite3.connect("books.db")
   cur=conn.cursor()
   cur.execute("SELECT * FROM book WHERE title=? OR author=? OR year=? OR isbn=?", (title,author,year,isbn))
    rows=cur.fetchall()
    conn.close()
    return rows
```

```
def delete(id):
   conn=sqlite3.connect("books.db")
   cur=conn.cursor()
   cur.execute("DELETE FROM book WHERE id=?",(id,))
   conn.commit()
   conn.close()

def update(id,title,author,year,isbn):
   conn=sqlite3.connect("books.db")
   cur=conn.cursor()
   cur.execute("UPDATE book SET title=?, author=?, year=?, isbn=? WHERE
id=?",(title,author,year,isbn,id))
   conn.commit()
   conn.close()

connect()
#insert("The Sun","John Smith",1918,913123132)
#delete(3)
#update(4,"The moon","John Smooth",1917,99999)
#print(view())
#print(search(author="John Smooth"))
```