

## DAILY ASSESSMENT FORMAT

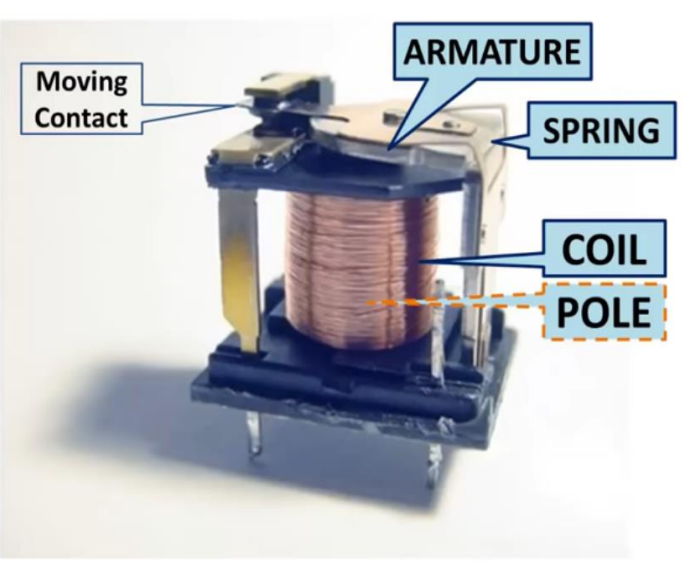
Date:	30-05-2020	Name:	BINDUSHRI
Course:	Logic designs	USN:	4AL17EC011
Topic:	1.applications of programmable logic controller.		6 <sup>th</sup> A
Github Repository:	Bindushri		

### FORENOON SESSION DETAILS

YouTube <sup>IN</sup>

Search

Q



Helping

Get Grammarly

Ad

www.grammarly.co

Up next

How could this be done with a normally closed PBT?

1:34:34

Th

Th

Da

2.6

41:29

Robust Modal

Decompositions

for Fluid Flows

11:35

Mk

Ear

1.3

15:59

What is a PLC? PLC Basics Pt1

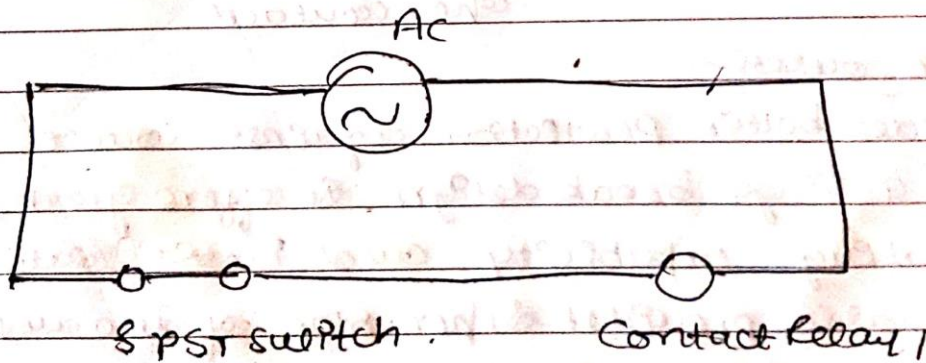
1,658,555 views • Nov 5, 2012

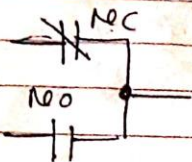
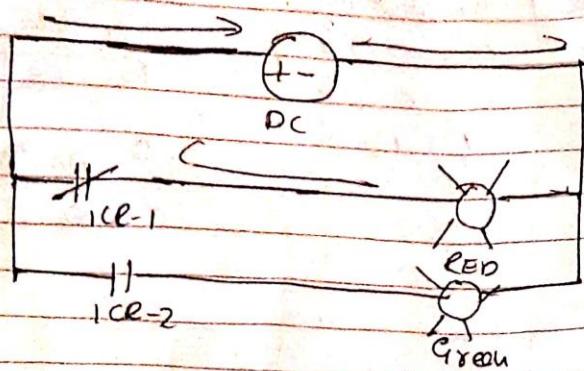
8.7K
407
SHARE
SAVE
...

Eng 3 - 30-05-2020.

## Applications of Programmable Logic Controller

- Operator Interface
- SPST Switch
- Red Indicator
- Green Indicator
- Power Source
- Alternating current
- Direct current



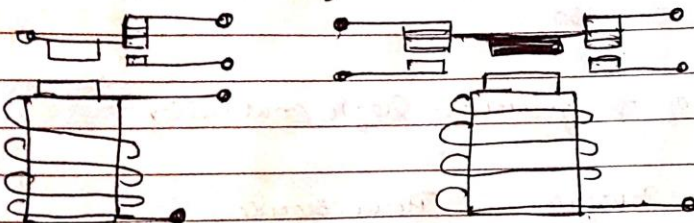


Normally  
closed  
contact



Normally  
open contact.

It is thus for a small transmission 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 contact

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 arcing contacts by more than 50% compared to a single break design.

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

Name:Bindushri  
USN:4AL17EC011  
Sem&Sec:6<sup>th</sup> A

## AFTERNOON SESSION DETAILS

### Image of session

→ [udemy.com/course/the-python-mega-course/learn/lecture/20104824#overview](https://udemy.com/course/the-python-mega-course/learn/lecture/20104824#overview)

**Udemy** The Python Mega Course: Build 10 Real World Applications

Your progress ▾ Share

**Course content**

- ☒ 199. Creating a "Login Page" (Frontend) 22min
- ☒ 200. Creating a "Sign Up Page" (Frontend) for New Users 10min
- ☒ 201. Getting User Input 6min
- ☒ 202. Implementing the "Sign Up Page" (Backend) 11min
- ☐ 203. Creating a "Sign Up Success Page" (Frontend) 4min
- ☐ 204. Switching Pages 3min
- ☐ 205. Implementing the "Login Page" (Backend) 17min
- ☐ 206. Displaying Text to the User 15min

**About this course**

Overview Q&A Notes Announcements

3:07 / 4:27

virtual keyboard not allowed, single mode, not docked  
Provider: sdl2  
NVIDIA texture support is available  
Start application main loop

```
>>> datetime.now()
datetime.datetime(2019, 11, 11, 11, 19, 17, 969847)
>>> datetime.now().strftime("%Y-%m-%d %H-%M-%S")
2019-11-11 11-19-17
```



20-05-2020.

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

Apperception: Build a webcam Motion Detector.

import cv2, time

from datetime import datetime

first\_frame = None

status = 0

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)

    (c, 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(counts)
- cv2.rectangle(frame, (x, y), (x+w, y+h),
                  (0, 255, 0), 3)
- status_lst.append(status)
```

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

```
if status_lst[-1] == 0 and status_lst[-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'):
```

```
- break if status == 1:
```

```
    break
    times.append(datetime.now())
```

```
print(status_lst)
```

```
print(times)
```

```
del times
```

```
del status_lst
```

```
for i in range(0, len(times), 2):
```

```
    df = df.append({"start": times[i], "end": times[i+1]})
```

```
del
```

```
ignore_index=True
```

```
df.to_csv("Times.csv")
```

```
Video.release()
```

```
cv2.destroyAllWindows
```



## Excercise program:

The screenshot displays a Visual Studio Code workspace titled "acc.py - Untitled (Workspace) - Visual Studio Code". The Explorer panel on the left shows the project structure with three groups: GROUP 1 (acc.py account), GROUP 2 (balance.txt account), and GROUP 3 (backend.py 1). The main editor area shows three files: acc.py, balance.txt, and backend.py.

**acc.py**

```
1 class Account:
2
3
4     def __init__(self,filepath):
5         self.filepath=filepath
6         with open(filepath,'r') as file:
7             self.balance=int(file.read())
8
9
10    def withdraw(self,ammount):
11        self.balance=self.balance-ammount
12
13
14    def deposit(self,ammount):
15        self.balance=self.balance+ammount
16
17
18    def commit(self):
19        with open(self.filepath,'w') as fi
20            file.write(str(self.balance))
21
22
23 account=Account("account//balance.txt")
24 print(account.balance)
25 account.deposit(100)
26 print(account.balance)
27 account.commit()
```

**balance.txt**

```
1 1200
```

**backend.py**

```
1 import sqlite3
2 class Database:
3
4     def __init__(self,db):
5         self.conn=sqlite3.connect(db)
6         self.cur=self.conn.cursor()
7         self.cur.execute("CREATE TABLE IF NOT EXISTS book (id I
8         self.conn.commit()
9
10
11     def insert(self,title,author,year,isbn):
12         self.cur.execute("INSERT INTO book VALUES (NULL,?,?,?,
13         self.conn.commit()
14
15
16     def view(self):
17         self.cur.execute("SELECT * FROM book")
18         rows=self.cur.fetchall()
19         return rows
20
21
22     def search(self,title="",author="",year="",isbn=""):
23         self.cur.execute("SELECT * FROM book WHERE title=? OR a
24         rows=self.cur.fetchall()
25         return rows
26
27     def delete(self,id):
28         self.cur.execute("DELETE FROM book WHERE id=?", (id,))
29         self.conn.commit()
```

The terminal at the bottom shows the following output:

```
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>
```

The status bar at the bottom indicates the Python version is 3.8.3 32-bit, the file encoding is UTF-8, and the line and column numbers are 25, 16.

The image displays a Visual Studio Code workspace with two main sections. The top section shows the development of a bank account system, and the bottom section shows the development of a book store application.

### Top Section: Bank Account System

**Explorer:** Shows files including `acc.py`, `balance.txt`, `backend.py`, `books.db`, and `frontend.py`.

**acc.py:**

```

1 class Account:
2
3
4
5     def __init__(self,filepath):
6         self.filepath=filepath
7         with open(filepath,'r') as file:
8             self.balance=int(file.read())
9
10
11     def withdraw(self,ammount):
12         self.balance=self.balance-ammount
13
14
15     def deposit(self,ammount):
16         self.balance=self.balance+ammount
17
18
19     def commit(self):
20         with open(self.filepath,'w') as fi
21             file.write(str(self.balance))
22
23
24 account=Account("account//balance.txt")
25 print(account.balance)
26 account.withdraw(300)
27 print(account.balance)
28 account.commit()

```

**balance.txt:**

```

1 900

```

**backend.py:**

```

1 import sqlite3
2 class Database:
3
4     def __init__(self,db):
5         self.conn=sqlite3.connect(db)
6         self.cur=self.conn.cursor()
7         self.cur.execute("CREATE TABLE IF NOT EXISTS book (id I
8         self.conn.commit()
9
10
11     def insert(self,title,author,year,isbn):
12         self.cur.execute("INSERT INTO book VALUES (NULL,?,?,?,?
13         self.conn.commit()
14
15
16     def view(self):
17         self.cur.execute("SELECT * FROM book")
18         rows=self.cur.fetchall()
19         return rows
20
21
22     def search(self,title="",author="",year="",isbn=""):
23         self.cur.execute("SELECT * FROM book WHERE title=? OR a
24         rows=self.cur.fetchall()
25         return rows
26
27     def delete(self,id):
28         self.cur.execute("DELETE FROM book WHERE id=?", (id,))
29         self.conn.commit()

```

**Terminal:**

```

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>

```

### Bottom Section: BookStore Application

**Explorer:** Shows files including `frontend.py`, `backend.py`, and `books.db`.

**frontend.py:**

```

1 from tkinter import *
2 import backend
3
4 def get_selected_row(event):
5     global selected_tuple
6     index=list1.curselection()[0]
7     selected_tuple=list1.get(index)
8     e1.delete(0,END)
9     e1.insert(END,selected_tuple[1])
10    e2.delete(0,END)
11    e2.insert(END,selected_tuple[2])
12    e3.delete(0,END)
13    e3.insert(END,selected_tuple[3])
14    e4.delete(0,END)
15    e4.insert(END,selected_tuple[4])
16
17 def view_command():
18     list1.delete(0,END)
19     for row in backend.view():
20         list1.insert(END,row)
21
22 def search_command():
23     list1.delete(0,END)
24     for row in backend.search(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()):
25         list1.insert(END,row)
26
27 def add_command():
28     backend.insert(title_text.get(),author_text.get(),year_text.get(),isbn_text.get())
29     list1.delete(0,END)
30     list1.insert(END,(title_text.get(),author_text.get(),year_text.get(),isbn_text.get()))

```

**Terminal:**

```

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

```

**BookStore Application Window:**

The application window titled "BookStore" contains a table with columns: Title, Year, Author, and ISBN. The table displays the following data:

Title	Year	Author	ISBN
the sea	1918	john tablet	913123132

Buttons on the right side of the window include: View all, Search entry, Add entry, Update selected, Delete selected, and Close.

## 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",1917,999999)
#print(view())
#print(search(author="John Smooth"))
```





