DAILY ASSESSMENT

Date:	**29/05/2020**	**Name:**	**CHANDANA.R**
Course:	Logic design	**USN:**	**4AL16EC017**
Topic:	Analysis of clocked sequential circuits, Digital clock design	**Semester & Section:**	**8(A)**
Github Repository:	**chandana-shaiva**		

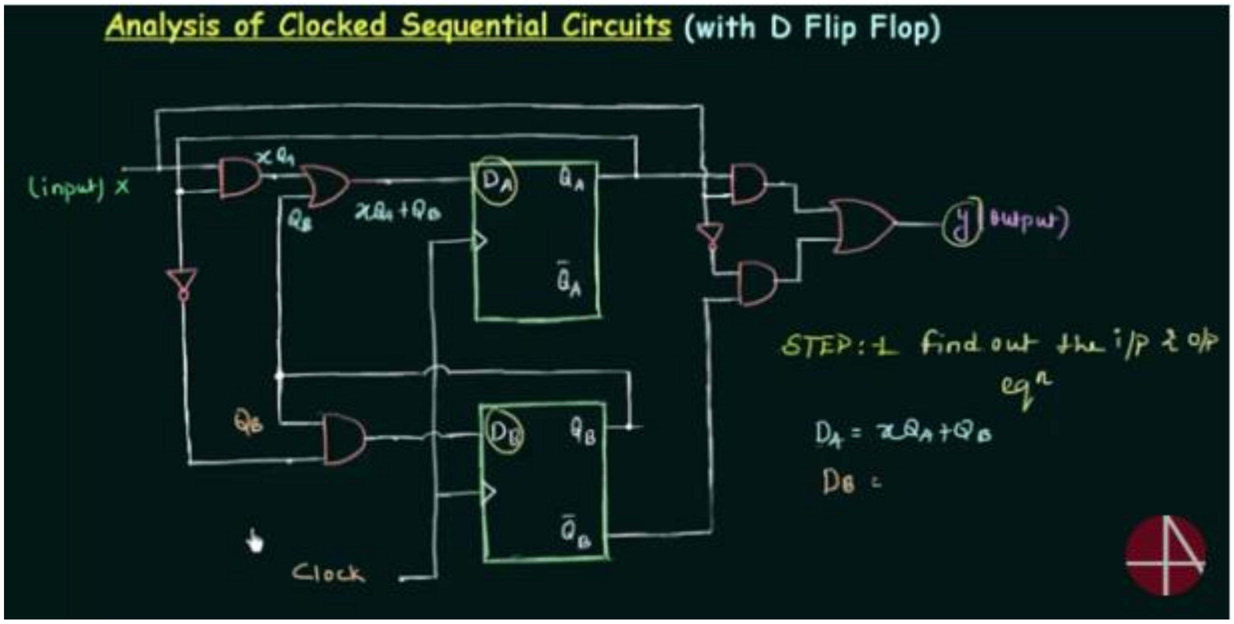
FORENOON SESSION DETAILS

REPORT:

- **Analysis of Clocked Sequential Circuits (with D Flip Flop)**

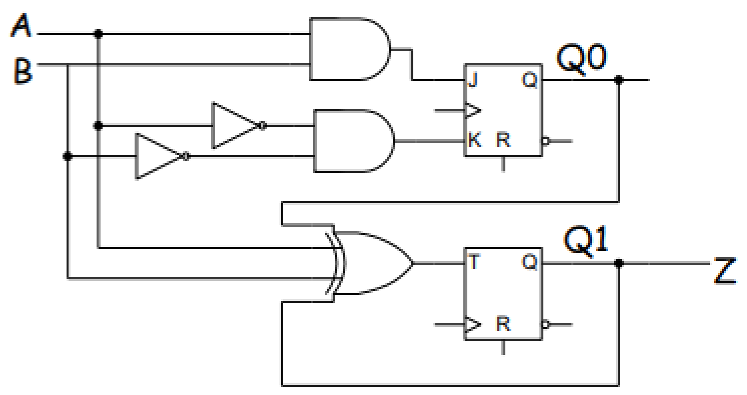
- The behavior of a clocked sequential circuit is determined from its inputs, outputs and state of the flip-flops (i.e., the output of the flip-flops).
- The analysis of a clocked sequential circuit consists of obtaining a table of a diagramo of the time sequences of inputs, outputs and states.

Consider the following circuit


Analysis of Clocked Sequential Circuits (with D Flip Flop)
(input) x
xQA
QB
xQA+QB
DA
QA
Q̄A
QB
DB
QB
Q̄B
Clock
y (output)
STEP: 1 find out the i/p & o/p eq^n

$D_A = xQ_A + Q_B$

$D_B =$



The circuit has two inputs A and B, one output Z, The circuit has two flip-flops (different types) with outputs Q0 and Q1 (This implies that there are as many as 4 different states in the circuit, namely Q0Q1 = 00, 01, 11, 10). The circuit output depends on the current state (flip-flop outputs) only.

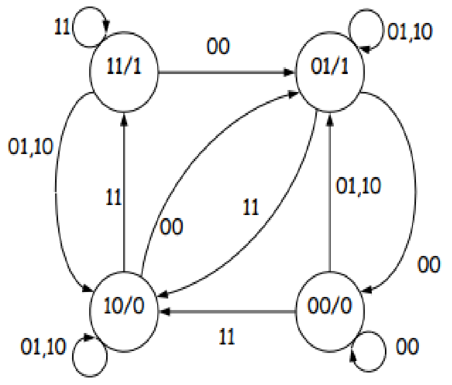
$$Z = Q_1$$

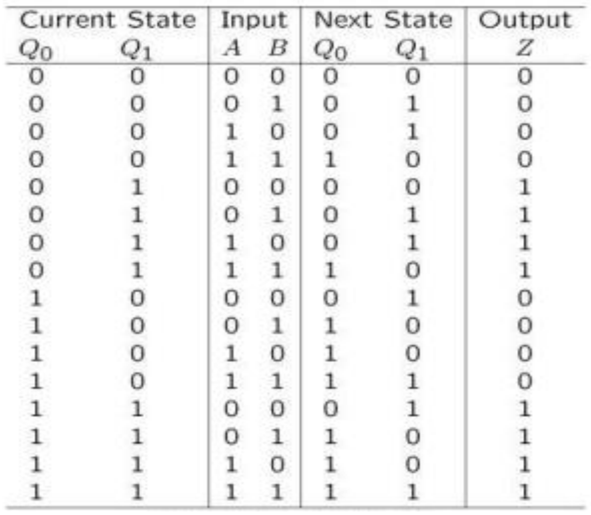
$$J_0 = AB$$
$$K_0 = A'B'$$

$$T_1 = A \oplus B \oplus Q_0 \oplus Q_1$$

Current State	J_0K_0				T_1			
Q_0Q_1	$AB = 00$	01	10	11	$AB = 00$	01	10	11
00	01	00	00	10	0	1	1	0
01	01	00	00	10	1	0	0	1
10	01	00	00	10	1	0	0	1
11	01	00	00	10	0	1	1	0

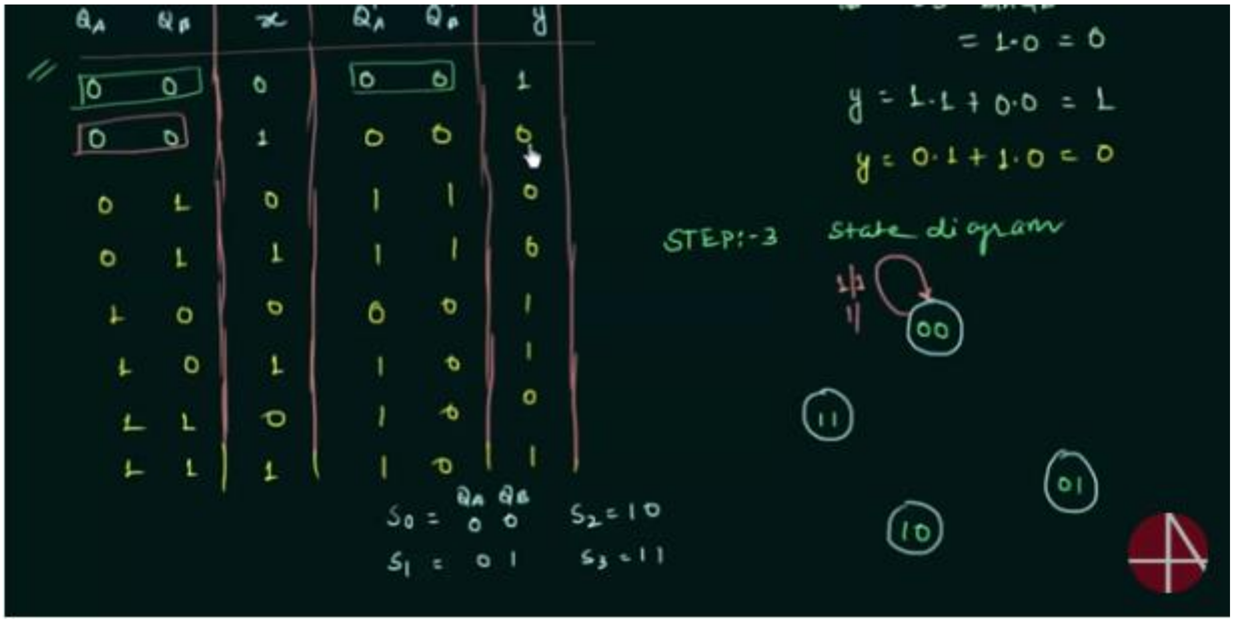
Another way to illustrate the behavior of a clocked sequential circuit is with a state diagram.





Current State		Input		Next State		Output
Q_0	Q_1	A	B	Q_0	Q_1	Z
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	1
1	0	0	0	0	1	0
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

Each “bubble” (state bubble) in the state diagram represents one state of the system. The flip-flop outputs that correspond to that state are labeled inside of theν bubble. Each edge leaving a bubble represents a possible transition to another state onceο the active clock edge arrives. The edges are labeled with the input values that cause the transition to occur.ν In this state diagram, the output values are labeled inside of the state bubbles.ο



Q_A	Q_B	x	Q_A'	Q_B'	y
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	1	0	0
1	1	1	1	0	1

$= 1\cdot 0 = 0$

$y = 1\cdot 1 + 0\cdot 0 = 1$

$y = 0\cdot 1 + 1\cdot 0 = 0$

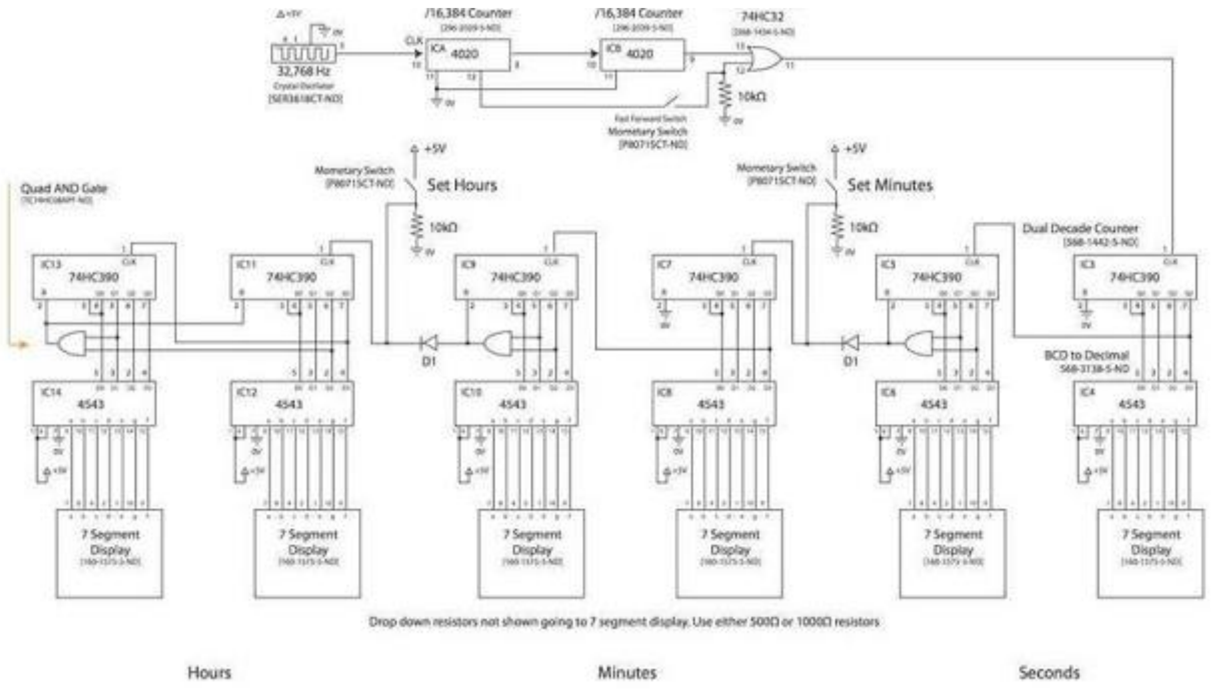
$S_0 = \overset{Q_A}{0}\ \overset{Q_B}{0}$ $\quad S_2 = 10$

$S_1 = 01$ $\quad S_3 = 11$

➢ Digital clock design:

- There are many stuff on the internet about this project, but we are gonna add something or two.

- This circuit was a school project and it was a 24h clock, but i decided to extend it to 12h and 24h with the transferring between them using a switch so you can choose whatever mode you want.
- Anyway, i made this circuit just for fun so i just simulated it on proteus, i.e we're not going into pcb design.



The main parts of the circuit are as follows:

1- **Timer 555**: Responsible for generating the clock pulses for the counters, the frequency of the output shoul be 1 hz which means 1 second for each pulse.

2- **Counters**: Responsible for generating the time in BCD (Binary Coded decimal).

3- **Decoders** : Takes the BCD of the counter as input and produces 7 segment output .

4- 7 **segments** : Displays the time, of course

MSB---LSB
0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101

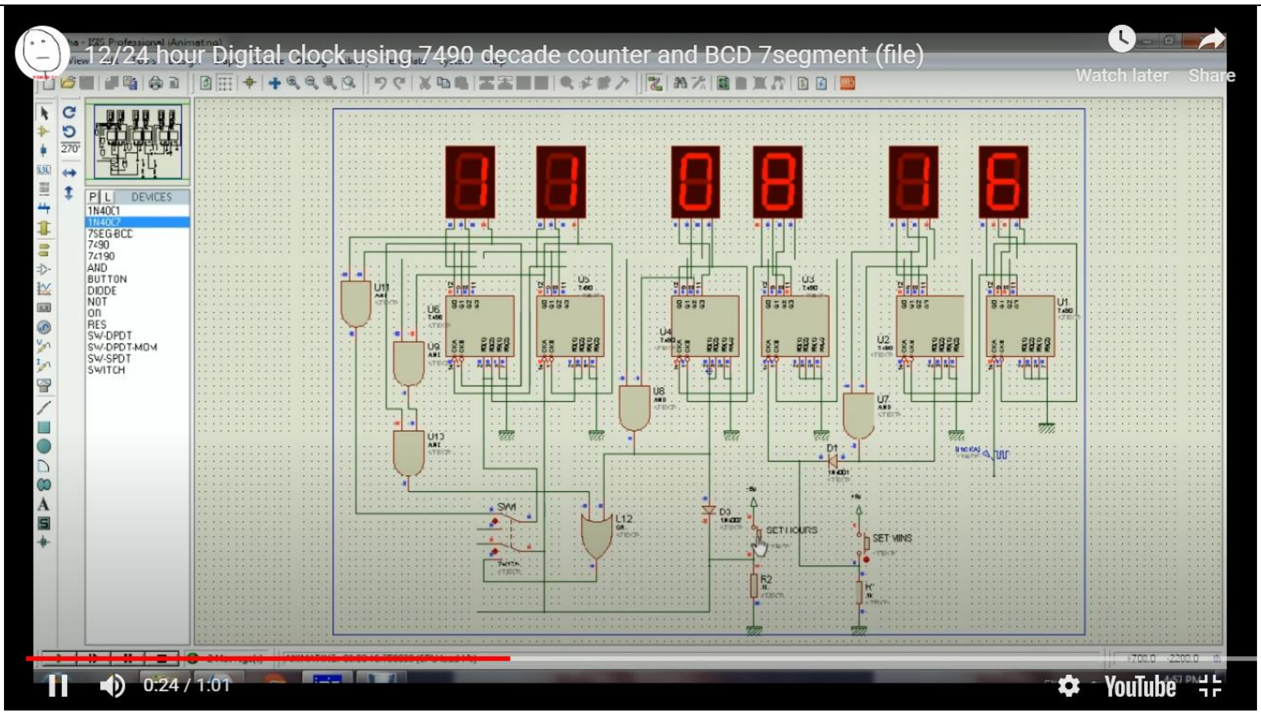
6: 0110
7: 0111
8: 1000
9: 1001
0:   0000

Remember that 7490 decade counters respond only to the pulses that go from 1 to 0 and notice that this case only happens in the BCD code above when the output changes from 9 to 0 (the Most significant bit changes from 1 to 0). So, we'll just connect the clock input of the 2nd counter to the most significant bit of the output of the first counter.
The 4th counter will be the same as the second one so we are clocking it using the Most Significant Bit of the output of the previous one.

Again, the 5th counter is the same as the 3rd one and takes its clock from the AND gate.

The 5th and the 6th counters are responsible for hours so they are limited to 23, and resets themselves to 00 when the 5th counter is 4 and the last one is 2 (24).
This is done using and gate with Q2 (3rd bit) of the 5th counter as one input and Q1 (second bit) of the last counter as the other input, and the output of this AND gate will be connected to both resets of the last 2 counters.

When the last counter is 0(0000) or 1(0001), Q1 which is one of the inputs to the AND gate will be 0 so the output of the AND gate will be zero. when it counts to 2 this bit will be 1 so the output of the and gate will depend on the the other input which is Q2 of the previous counter, and this bit will be zero until it reaches 4 (0100),So, the output of the and gate will be 1 (0--->1) resetting both counters to 00,

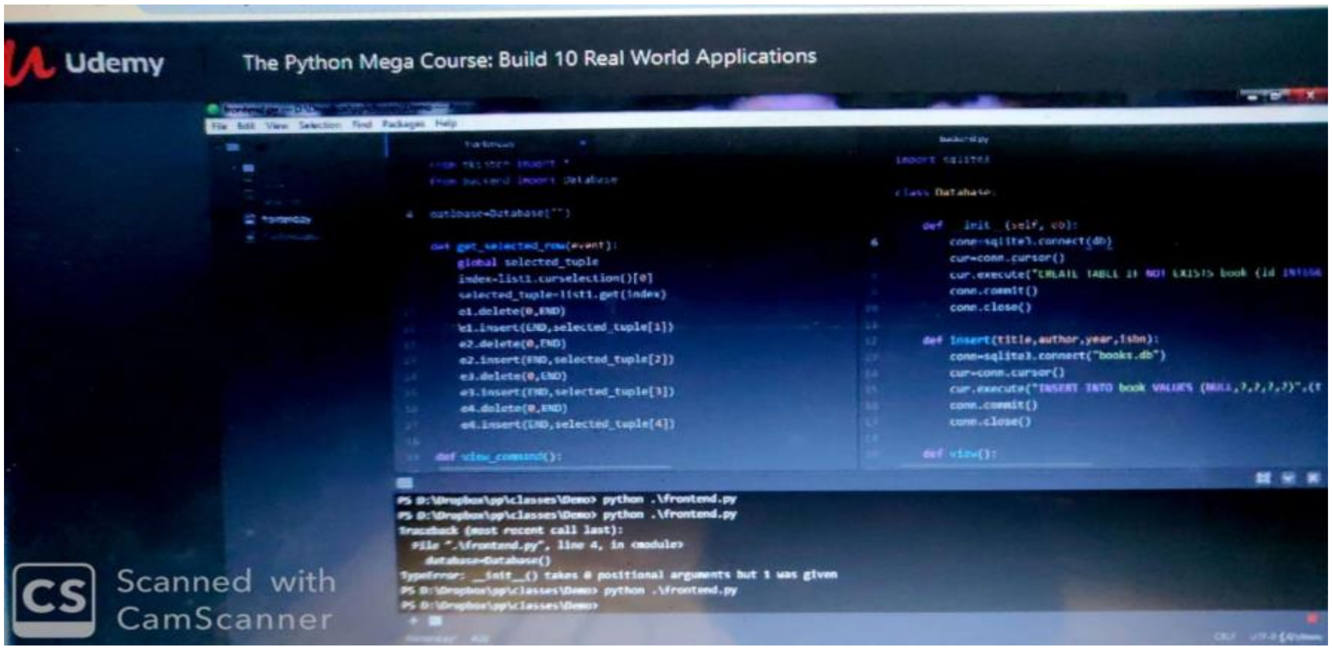

12/24 hour Digital clock using 7490 decade counter and BCD 7segment (file)
Watch later
Share
DEVICES
1N4001
1N4007
7SEG-BCD
7490
74190
AND
BUTTON
DIODE
NOT
OR
RES
SW-DPDT
SW-DPDT-MOM
SW-SPDT
SWITCH
U11
U6
U5
U4
U3
U2
U1
U9
U8
U7
U10
U12
SW1
D1
D3
R2
SET HOURS
SET MINS
0:24 / 1:01
YouTube

DAILY ASSIGEMENTS DETAILS:

Date:	**29/05/2020**	**Name:**	**Chandana.R**
Course:	Python	**USN:**	**4AL16EC017**
Topic:	Object oriented programming	**Semester & Section:**	**8(A)**
Github Repository:	**chandana-shaiva**		

AFTERNOON SESSION

Session image:



```
from tkinter import *
from backend import Database

database=Database("")

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():
```

```
import sqlite3

class Database:

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

    def view():
```

```
PS D:\Dropbox\pp\classes\Demo> python .\frontend.py
PS D:\Dropbox\pp\classes\Demo> python .\frontend.py
Traceback (most recent call last):
  File ".\frontend.py", line 4, in <module>
    database=Database()
TypeError: __init__() takes 0 positional arguments but 1 was given
PS D:\Dropbox\pp\classes\Demo> python .\frontend.py
PS D:\Dropbox\pp\classes\Demo>
```

REPORT:

- Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy.

Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class*

- The class has a documentation string, which can be accessed via *ClassName.__doc__*.
- The *class_suite* consists of all the component statements defining class members, data attributes and functions.

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its ***__init__*** method accepts.

Built-In Class Attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute −

- **__dict__** − Dictionary containing the class's namespace.
- **__doc__** − Class documentation string or none, if undefined.
- **__name__** − Class name.
- **__module__** − Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- **__bases__** − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list

Garbage Collection

Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection

Program:

```
from tkinter import *
from backend import Database

database=Database("books.db")

class Window(object):

def __init__(self,window):

self.window = window
self.window.wm_title("BookStore")
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)

self.title_text=StringVar()
        self.e1=Entry(window,textvariable=self.title_text)
        self.e1.grid(row=0,column=1)
self.author_text=StringVar()
        self.e2=Entry(window,textvariable=self.author_text)
        self.e2.grid(row=0,column=3)

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

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

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

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

        self.list1.configure(yscrollcommand=sb1.set)
        sb1.configure(command=self.list1.yview)
self.list1.bind('<<ListboxSelect>>',self.get_selected_row)

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

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

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

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

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

        b6=Button(window,text="Close", width=12,command=window.destroy)
        b6.grid(row=7,column=3)
defget_selected_row(self,event):
        index=self.list1.curselection()[0]
self.selected_tuple=self.list1.get(index)
        self.e1.delete(0,END)
self.e1.insert(END,self.selected_tuple[1])
        self.e2.delete(0,END)
        self.e2.insert(END,self.selected_tuple[2])
        self.e3.delete(0,END)
        self.e3.insert(END,self.selected_tuple[3])
        self.e4.delete(0,END)
        self.e4.insert(END,self.selected_tuple[4])

defview_command(self):
        self.list1.delete(0,END)
        for row in database.view():
            self.list1.insert(END,row)

defsearch_command(self):
        self.list1.delete(0,END)
        for row is
 database.search(self.title_text.get(),self.author_text.get(),self.year_text.get(),self.isbn_text.get()):
            self.list1.insert(END,row)

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

window=Tk()
Window(window)
window.mainloop()
```