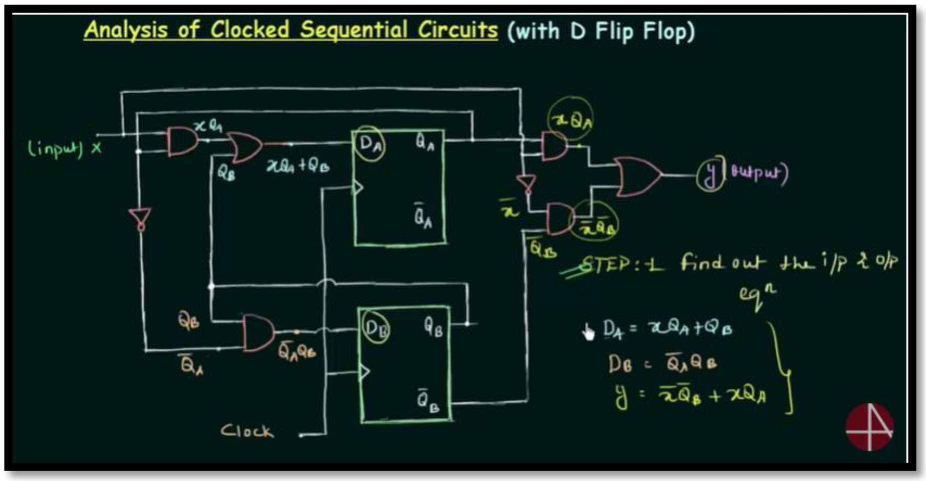
DAILY ASSESSMENT FORMAT

Date:	**29/05/2020**	**Name:**	**Davis S. Patel**
Course:	**Logic Design**	**USN:**	**4AL16EC045**
Topic:	**Analysis of clocked sequential Circuits** **Digital clock design**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

FORENOON SESSION DETAILS

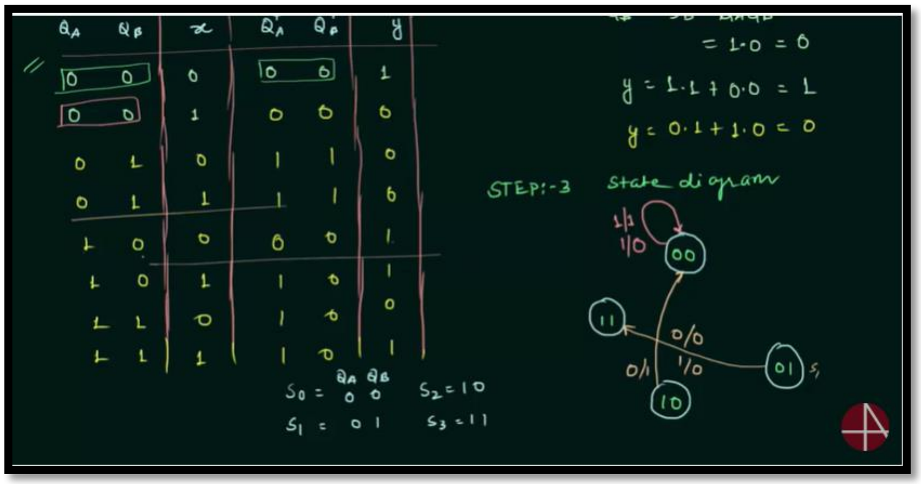
Image of session



Analysis of Clocked Sequential Circuits (with D Flip Flop)

STEP:-1 find out the i/p & o/p eqn

$$D_A = xQ_A + Q_B$$
$$D_B = \bar{Q}_AQ_B$$
$$y = \bar{x}\bar{Q}_B + xQ_A$$



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

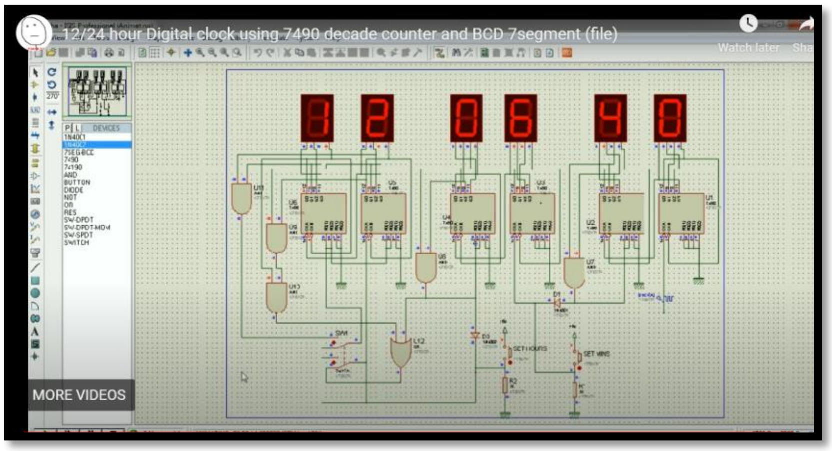
$= 1\cdot0 = 0$

$y = 1\cdot1 + 0\cdot0 = 1$

$y = 0\cdot1 + 1\cdot0 = 0$

STEP:-3 State diagram

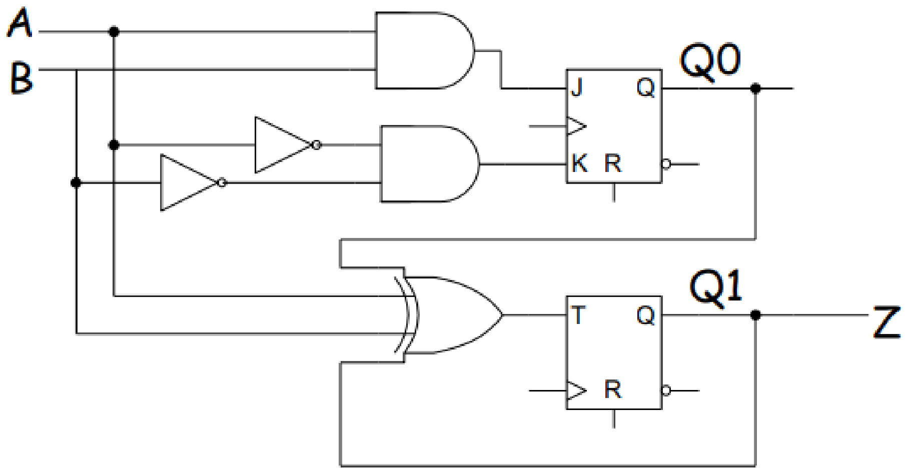
	Q_A Q_B		
S_0 =	0 0	S_2 = 10	
S_1 =	0 1	S_3 = 11	


12/24 hour Digital clock using 7490 decade counter and BCD 7segment (file)
Watch later
MORE VIDEOS

Report –

Analysis of clocked sequential Circuits

The analysis of a clocked sequential circuit consists of obtaining a table of a diagram of the time sequences of inputs, outputs and states. Consider the following circuit. We want to determine how it will behave.



The circuit has two inputs A and B, one output Z (it happens that the output is equal to one of the flip flop outputs).

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

J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$Q'(t)$

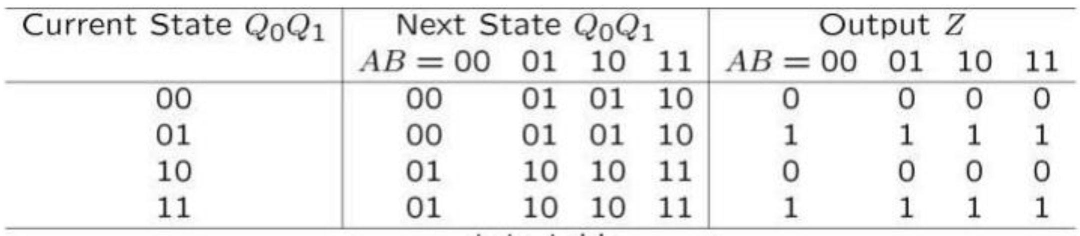
JKFF Behavior

T	$Q(t+1)$
0	$Q(t)$
1	$Q'(t)$

TFF Behavior

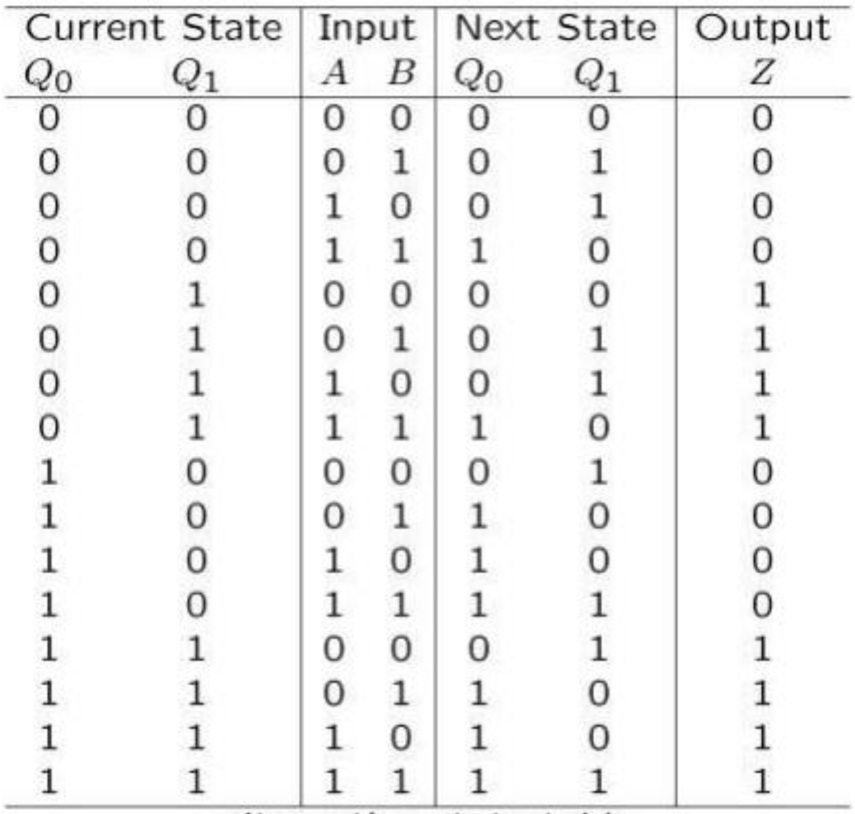
Current State Q_0Q_1	Next State Q_0				Next State Q_1			
	$AB = 00$	01	10	11	$AB = 00$	01	10	11
00	0	0	0	1	0	1	1	0
01	0	0	0	1	0	1	1	0
10	0	1	1	1	1	0	0	1
11	0	1	1	1	1	0	0	1

Current State Q_0Q_1	Next State Q_0Q_1				Output Z			
	$AB = 00$	01	10	11	$AB = 00$	01	10	11
00	00	01	01	10	0	0	0	0
01	00	01	01	10	1	1	1	1
10	01	10	10	11	0	0	0	0
11	01	10	10	11	1	1	1	1



Current State Q_0Q_1	Next State Q_0Q_1				Output Z			
	$AB = 00$	01	10	11	$AB = 00$	01	10	11
00	00	01	01	10	0	0	0	0
01	00	01	01	10	1	1	1	1
10	01	10	10	11	0	0	0	0
11	01	10	10	11	1	1	1	1

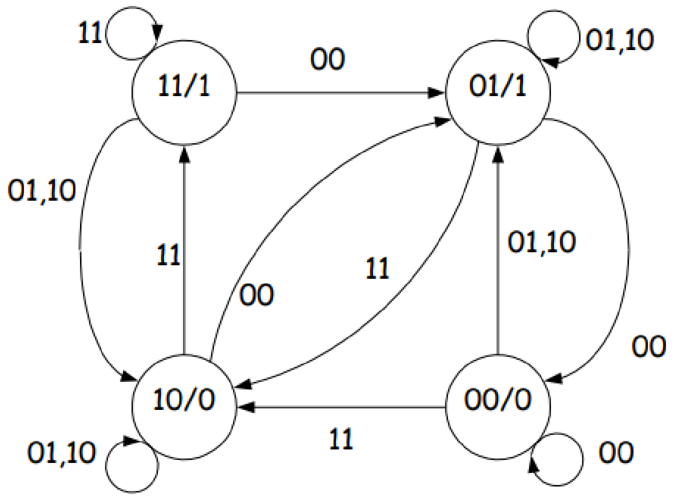
The output Z (in this example) is only a function of the current state; it does not depend on the inputs.



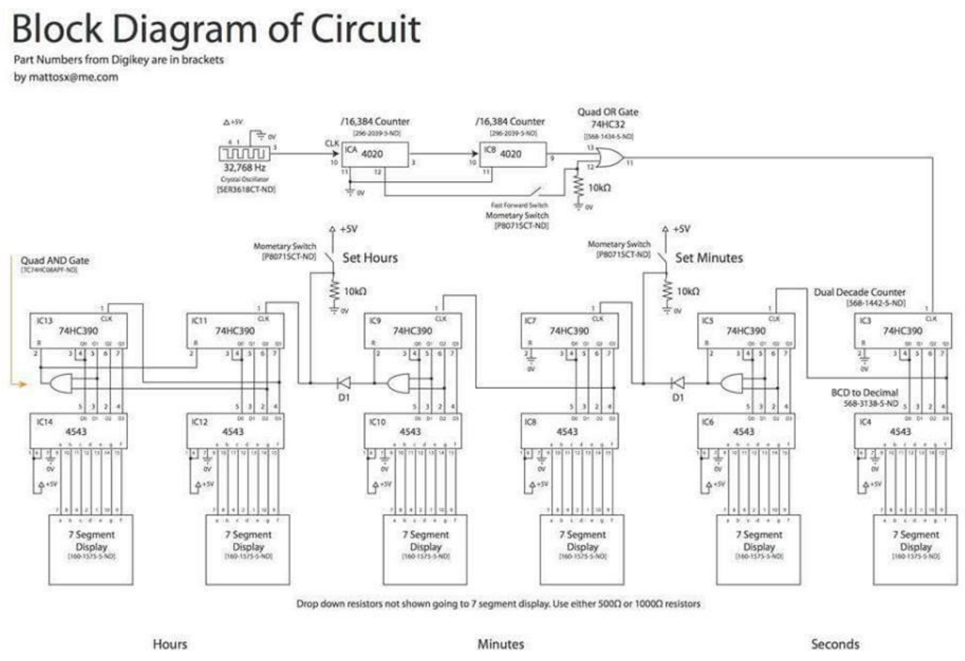
Current State		Input		Next State		Output
Q_0	Q_1	A	B	Q_0	Q_1	Z
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	1
1	0	0	0	0	1	0
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

alternative state table

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



Digital clock 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 should 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.

The circuit works as follows:

555 timer produces 1 second pulses to the clock input of the first counter which is responsible the first column of seconds, so its output will change every second.

The counter produces numbers from 0 to 9 in BCD form and automatically resets to 0 after that.
So the output of the first counter will count from 0 to 9 every second.

The output of the counters are named: Q0, Q1, Q2, Q3

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 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.

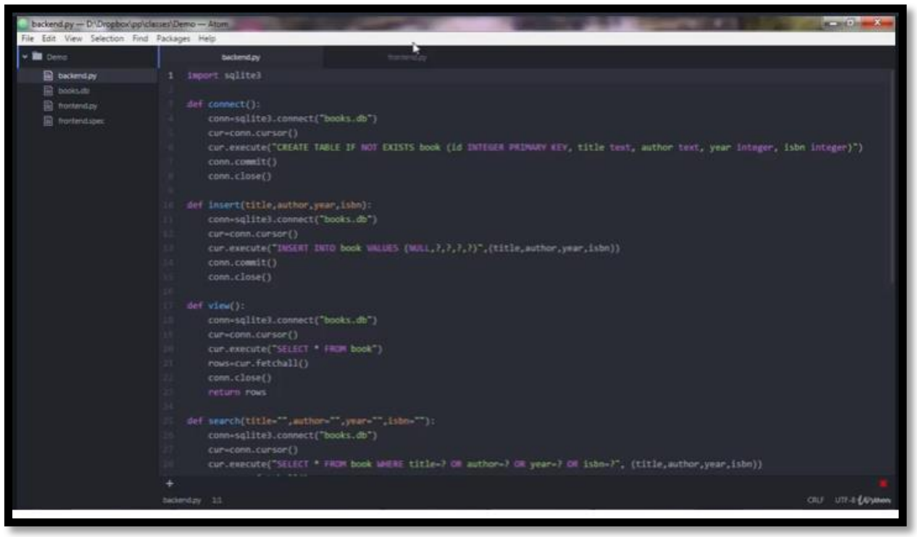
The output of these counters are converted to 7 segment output using 7447 decoders, then to the 7 segment.

DAILY ASSESSMENT FORMAT

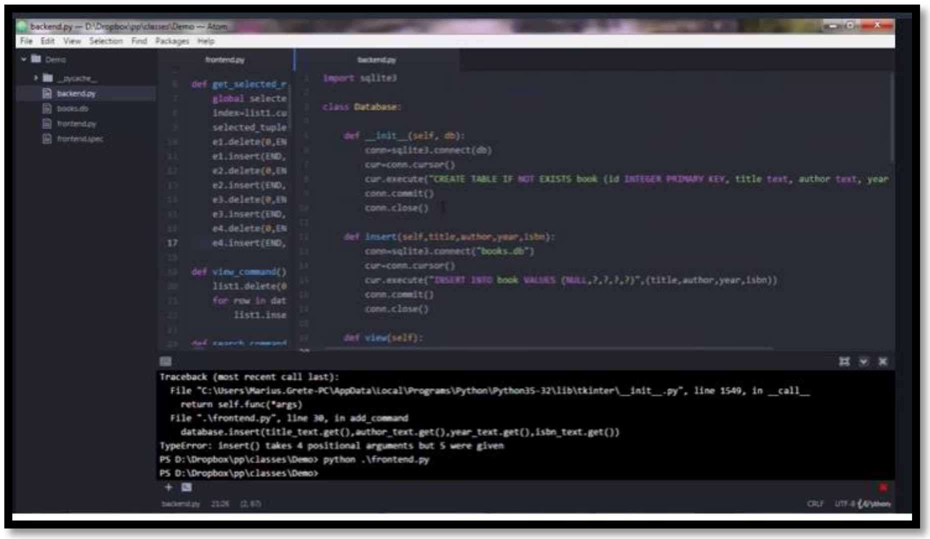
Date:	**29/05/2020**	**Name:**	**Davis S. Patel**
Course:	**Python Course**	**USN:**	**4AL16EC045**
Topic:	**Object Oriented Programming**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

AFTERNOON SESSION DETAILS

Image of Session



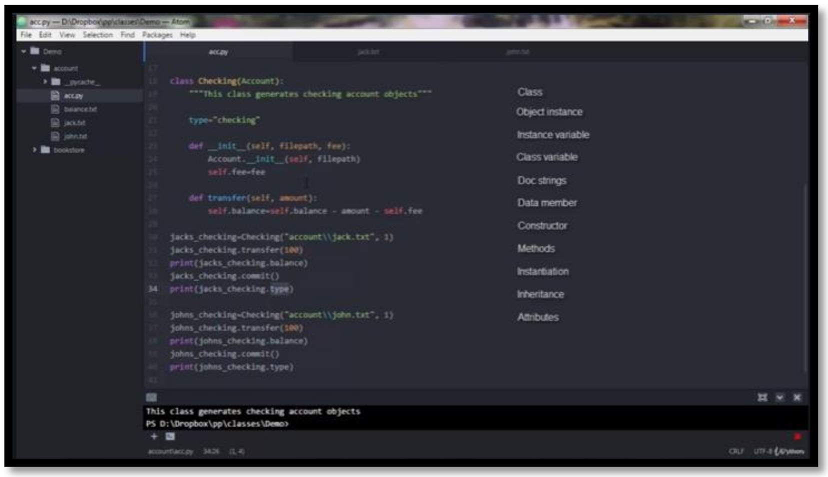
```
import sqlite3

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()

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))
```



```
def get_selected_r
    global selecte
    index=list1.cu
    selected_tuple
    e1.delete(0,EN
    e1.insert(END,
    e2.delete(0,EN
    e2.insert(END,
    e3.delete(0,EN
    e3.insert(END,
    e4.delete(0,EN
    e4.insert(END,

def view_command()
    list1.delete(0
    for row in dat
        list1.inse
```

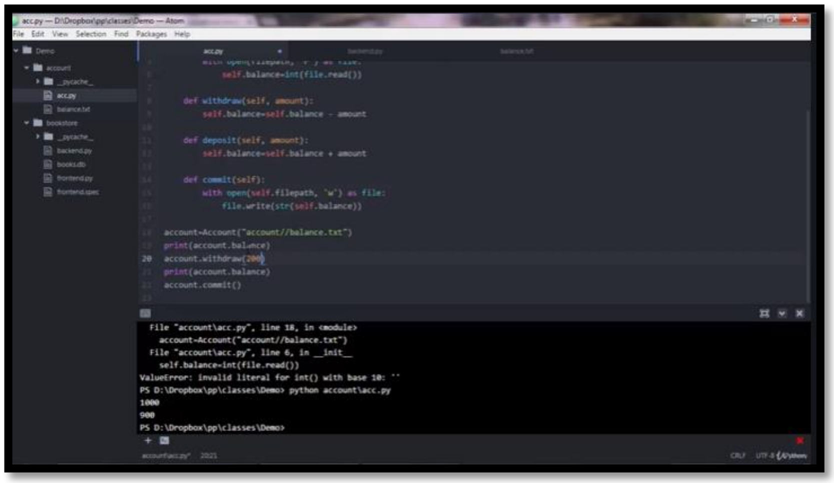
```
import sqlite3

class Database:

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

    def insert(self,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()

    def view(self):
```

```
Traceback (most recent call last):
  File "C:\Users\Marius.Grete-PC\AppData\Local\Programs\Python\Python35-32\lib\tkinter\__init__.py", line 1549, in __call__
    return self.func(*args)
  File ".\frontend.py", line 30, in add_command
    database.insert(title_text.get(),author_text.get(),year_text.get(),isbn_text.get())
TypeError: insert() takes 4 positional arguments but 5 were given
PS D:\Dropbox\pp\classes\Demo> python .\frontend.py
PS D:\Dropbox\pp\classes\Demo>
```



```python
class Checking(Account):
    """This class generates checking account objects"""

    type="checking"

    def __init__(self, filepath, fee):
        Account.__init__(self, filepath)
        self.fee=fee

    def transfer(self, amount):
        self.balance=self.balance - amount - self.fee

jacks_checking=Checking("account\\jack.txt", 1)
jacks_checking.transfer(100)
print(jacks_checking.balance)
jacks_checking.commit()
print(jacks_checking.type)

johns_checking=Checking("account\\john.txt", 1)
johns_checking.transfer(100)
print(johns_checking.balance)
johns_checking.commit()
print(johns_checking.type)
```

```
This class generates checking account objects
PS D:\Dropbox\pp\classes\Demo>
```



```python
            self.balance=int(file.read())

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

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

    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(200)
print(account.balance)
account.commit()
```

```
  File "account\acc.py", line 18, in <module>
    account=Account("account//balance.txt")
  File "account\acc.py", line 6, in __init__
    self.balance=int(file.read())
ValueError: invalid literal for int() with base 10: ''
PS D:\Dropbox\pp\classes\Demo> python account\acc.py
1000
900
PS D:\Dropbox\pp\classes\Demo>
```

Report –

Object Oriented Programming

Object-oriented Programming, or OOP for short, is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.

The primitive data structures available in Python, like numbers, strings, and lists are designed to represent simple things like the cost of something, the name of a poem, and your favorite colors, respectively. Classes are used to create new user-defined data structures that contain arbitrary information about something. In the case of an animal, we could create an Animal () class to track properties about the Animal like the name and age.

Defining a class is simple in Python:

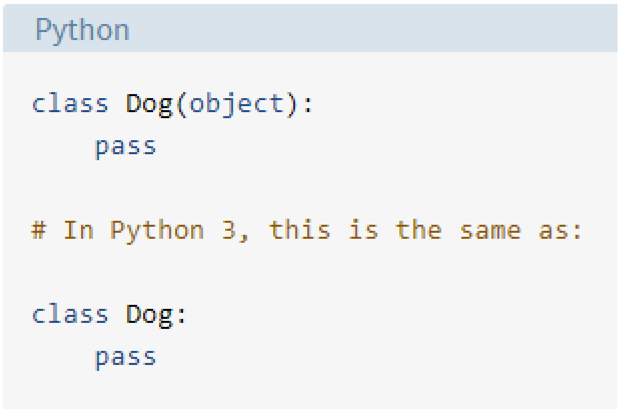
Python

```
class Dog:
    pass
```

Python Object Inheritance

Inheritance is the process by which one class takes on the attributes and methods of another. Newly formed classes are called child classes, and the classes that child classes are derived from are called parent classes.

It's important to note that child classes override or extend the functionality (e.g., attributes and behaviors) of parent classes. In other words, child classes inherit all of the parent's attributes and behaviors but can also specify different behavior to follow. The most basic type of class is an object, which generally all other classes inherit as their parent.


Python

```python
class Dog(object):
    pass

# In Python 3, this is the same as:

class Dog:
    pass
```

The *frontend.py* and *backend.py* scripts in OOP style –

#frontend.py

```python
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")


        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)


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)
```

```
    def get_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])



    def view_command(self):

        self.list1.delete(0,END)

        for row in database.view():

            self.list1.insert(END,row)



    def search_command(self):

        self.list1.delete(0,END)

        for row in database.search(self.title_text.get(),self.author_text.get(),self.year_text.get
(),self.isbn_text.get()):

            self.list1.insert(END,row)



    def add_command(self):

        database.insert(self.title_text.get(),self.author_text.get(),self.year_text.get(),self.isb
n_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()))

    def delete_command(self):

        database.delete(self.selected_tuple[0])

    def update_command(self):

        database.update(self.selected_tuple[0],self.title_text.get(),self.author_text.get(),self.year_text.get(),self.isbn_text.get())

window=Tk()

Window(window)

window.mainloop()
```

#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=self.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()

    def __del__(self):

        self.conn.close()
```

ECE-4year-Code-Challenge 2

MATLAB CODE FOR ECG:

```
x=0.01:0.01:2;
default=input(&#39;Press 1 if u want default ecg signal else press 2:\n&#39;);
if(default==1)
li=30/72;
a_pwav=0.25;
d_pwav=0.09;
t_pwav=0.16;
a_qwav=0.025;
d_qwav=0.066;
t_qwav=0.166;
a_qrswav=1.6;
d_qrswav=0.11;
a_swav=0.25;
d_swav=0.066;
t_swav=0.09;
a_twav=0.35;
d_twav=0.142;
t_twav=0.2;
a_uwav=0.035;
d_uwav=0.0476;
t_uwav=0.433;
else
rate=input(&#39;\n\nenter the heart beat rate :&#39;);
li=30/rate;
```

```
%p wave specifications
fprintf(&#39;\n\np wave specifications\n&#39;);
d=input(&#39;Enter 1 for default specification else press 2: \n&#39;);
if(d==1)
a_pwav=0.25;
d_pwav=0.09;
t_pwav=0.16;
else
a_pwav=input(&#39;amplitude = &#39;);
d_pwav=input(&#39;duration = &#39;);
t_pwav=input(&#39;p-r interval = &#39;);
d=0;
end

%q wave specifications
fprintf(&#39;\n\nq wave specifications\n&#39;);
d=input(&#39;Enter 1 for default specification else press 2: \n&#39;);
if(d==1)
a_qwav=0.025;
d_qwav=0.066;
t_qwav=0.166;
else
a_qwav=input(&#39;amplitude = &#39;);

d_qwav=input(&#39;duration = &#39;);
t_qwav=0.166;
d=0;
end
```

```
%qrs wave specifications

fprintf(&#39;\n\nqrs wave specifications\n&#39;);

d=input(&#39;Enter 1 for default specification else press 2: \n&#39;);

if(d==1)

a_qrswav=1.6;

d_qrswav=0.11;

else

a_qrswav=input(&#39;amplitude = &#39;);

d_qrswav=input(&#39;duration = &#39;);

d=0;

end


%s wave specifications

fprintf(&#39;\n\ns wave specifications\n&#39;);

d=input(&#39;Enter 1 for default specification else press 2: \n&#39;);

if(d==1)

a_swav=0.25;

d_swav=0.066;

t_swav=0.09;

else

a_swav=input(&#39;amplitude = &#39;);

d_swav=input(&#39;duration = &#39;);

t_swav=0.09;

d=0;

end


%t wave specifications
```

```
fprintf(&#39;\n\nt wave specifications\n&#39;);
d=input(&#39;Enter 1 for default specification else press 2: \n&#39;);
if(d==1)
a_twav=0.35;
d_twav=0.142;
t_twav=0.2;
else
a_twav=input(&#39;amplitude = &#39;);
d_twav=input(&#39;duration = &#39;);
t_twav=input(&#39;s-t interval = &#39;);
d=0;
end

%u wave specifications
fprintf(&#39;\n\nu wave specifications\n&#39;);
d=input(&#39;Enter 1 for default specification else press 2: \n&#39;);
if(d==1)
a_uwav=0.035;
d_uwav=0.0476;
t_uwav=0.433;
else

a_uwav=input(&#39;amplitude = &#39;);
d_uwav=input(&#39;duration = &#39;);
t_uwav=0.433;
d=0;
end
```

```
end
pwav=p_wav(x,a_pwav,d_pwav,t_pwav,li);
%qwav output
qwav=q_wav(x,a_qwav,d_qwav,t_qwav,li);
%qrswav output
qrswav=qrs_wav(x,a_qrswav,d_qrswav,li);
%swav output
swav=s_wav(x,a_swav,d_swav,t_swav,li);
%twav output
twav=t_wav(x,a_twav,d_twav,t_twav,li);
%uwav output
uwav=u_wav(x,a_uwav,d_uwav,t_uwav,li);
%ecg output
ecg=pwav+qrswav+twav+swav+qwav+uwav;
figure(1)
plot(x,ecg);
```