**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **28-05-2020** | **Name:** | **BHOOMIKA HEBBAR** |
| **Course:** | **Logic design** | **USN:** | **4AL17EC010** |
| **Topic:** | **Boolean equations for digital circuits. Combinational circuits: Conversion of MUX and Decoders to logic gates, design of 7 segment decoder with common anode display** | **Semester & Section:** | **6th & A** |
| **Github Repository:** | **bhoomika\_python** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**  **1.**    **2.**    **3.** |
| **Report – Report can be typed or hand written for up to two pages.**   1. **Boolean equations for digital circuits.** 2. **Combinational circuits:**  * **Conversion of Mux** * **Decoders to logic gates.**   **3. design of 7 segment decoder with common anode display**  **Boolean equations for digital circuits:** Boolean algebra is used to simplify Boolean expressions which represent combinational logic circuits. It reduces the original expression tequivalent expression that has fewer terms which means that less logic gates are needed to implement the combinational logic circuit. Laws of Boolean Algebra Boolean Algebra Laws are used to simplify boolean expressions. Basic Boolean Laws  1. Idempotent Law    * A \* A = A    * A + A = A 2. Associative Law    * (A \* B) \* C = A \* (B \* C)    * (A + B) + C = A + (B + C) 3. Commutative Law    * A \* B = B \* A    * A + B = B + A 4. Distributive Law    * A \* (B + C) = A \* B + A \* C    * A + (B \* C) = (A + B) \* (A + C) 5. Identity Law    * A \* 0 = 0     A \* 1 = A    * A + 1 = 1     A + 0 = A 6. Complement Law    * A \* ~A = 0    * A + ~A = 1 7. Involution Law    * ~(~A) = A 8. DeMorgan's Law    * ~(A \* B) = ~A + ~B    * ~(A + B) = ~A \* ~B  Redundancy Laws  1. Absorption    * A + (A \* B) = A    * A \* (A + B) = A 2. * (A \* B) + (A \* ~B) = A    * (A + B) \* (A + ~B) = A 3. * A + (~A \* B) = A + B    * A \* (~A + B) = A \* B   Each law is described by two parts that are duals of each other. The Principle of duality is   * Interchanging the + (OR) and \* (AND) operations of the expression. * Interchanging the 0 and 1 elements of the expression. * Not changing the form of the variables.  **DeMorgan’s Theorem** This theorem is useful in finding the **complement of Boolean function**. It states that the complement of logical OR of at least two Boolean variables is equal to the logical AND of each complemented variable.  DeMorgan’s theorem with 2 Boolean variables x and y can be represented as  x+yx+y’ = x’.y’  The dual of the above Boolean function is  x.yx.y’ = x’ + y’  Therefore, the complement of logical AND of two Boolean variables is equal to the logical OR of each complemented variable. Similarly, we can apply DeMorgan’s theorem for more than 2 Boolean variables also. Application of Boolean Algebra: [Combinational Logic Circuit Design](http://electronics-course.com/combinational-logic-design) comprises the following steps   1. From the design specification, obtain the truth table 2. From the truth table, derive the [Sum of Products](http://electronics-course.com/sum-of-products) Boolean Expression. 3. Use Boolean Algebra to simplify the boolean expression. The simpler the boolean expression, the less logic gates will be used. 4. Use [logic gates](http://electronics-course.com/logic-gates) to implement the simplified Boolean Expression.   **MUX:** In electronics, a multiplexer, also known as a data selector, is a device that selects between several analog or digital input signals and forwards it to a single output line. A multiplexer of inputs has select lines, which are used to select which input line to send to the output.  **DECODER TO LOGIC GATES:** These logic gates are the building blocks of combinational logic circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.  **Design of 7 segment decoder with common anode display**: A 7 segment LED display consists of an arrangement of 8 LEDs such that either all the anodes are common or cathodes are common. A common cathode 7 segment display consists of 8 pins – 7 input pins labeled from 'a' to 'g' and 8th pin as common ground pin.  A Digital Decoder IC, is a device which converts one digital format into another and one of the most commonly used devices for doing this is called the Binary Coded Decimal (BCD) to 7-Segment Display Decoder. ... In electronics there are two important types of 7-segment LED digital display. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **28-05-2020** | **Name:** | **BHOOMIKA HEBBAR** | |
| **Course:** | **UDEMY PYTHON MEGA\_COURSE** | **USN:** | **4AL17EC010** | |
| **Topic:** | **Object Oriented Programming** | **Semester & Section:** | **6th &A** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of the session:** | | | |
| **Report – Report can be typed or hand written for up to two pages.**  **Today I have learnt :**   1. Object Oriented Programming Explained 2. Turning this Application into OOP Style, Turning this Application into OOP Style,Part 2 3. Creating a Bank Account Object 4. Inheritance 5. OOP Glossary 6. GUI in OOP Design (Practice) 7. solution   **summary:**  Here are the *frontend.py*and *backend.py*scripts in OOP style. To execute this program you should execute the *frontend.py* file.   1. #frontend.py 2. from tkinter import \* 3. from backend import Database 5. database=Database("books.db") 7. class Window(object): 9. def \_\_init\_\_(self,window): 11. self.window = window 13. self.window.wm\_title("BookStore") 15. l1=Label(window,text="Title") 16. l1.grid(row=0,column=0) 18. l2=Label(window,text="Author") 19. l2.grid(row=0,column=2) 21. l3=Label(window,text="Year") 22. l3.grid(row=1,column=0) 24. l4=Label(window,text="ISBN") 25. l4.grid(row=1,column=2) 27. self.title\_text=StringVar() 28. self.e1=Entry(window,textvariable=self.title\_text) 29. self.e1.grid(row=0,column=1) 31. self.author\_text=StringVar() 32. self.e2=Entry(window,textvariable=self.author\_text) 33. self.e2.grid(row=0,column=3) 35. self.year\_text=StringVar() 36. self.e3=Entry(window,textvariable=self.year\_text) 37. self.e3.grid(row=1,column=1) 39. self.isbn\_text=StringVar() 40. self.e4=Entry(window,textvariable=self.isbn\_text) 41. self.e4.grid(row=1,column=3) 43. self.list1=Listbox(window, height=6,width=35) 44. self.list1.grid(row=2,column=0,rowspan=6,columnspan=2) 46. sb1=Scrollbar(window) 47. sb1.grid(row=2,column=2,rowspan=6) 49. self.list1.configure(yscrollcommand=sb1.set) 50. sb1.configure(command=self.list1.yview) 52. self.list1.bind('<<ListboxSelect>>',self.get\_selected\_row) 54. b1=Button(window,text="View all", width=12,command=self.view\_command) 55. b1.grid(row=2,column=3) 57. b2=Button(window,text="Search entry", width=12,command=self.search\_command) 58. b2.grid(row=3,column=3) 60. b3=Button(window,text="Add entry", width=12,command=self.add\_command) 61. b3.grid(row=4,column=3) 63. b4=Button(window,text="Update selected", width=12,command=self.update\_command) 64. b4.grid(row=5,column=3) 66. b5=Button(window,text="Delete selected", width=12,command=self.delete\_command) 67. b5.grid(row=6,column=3) 69. b6=Button(window,text="Close", width=12,command=window.destroy) 70. b6.grid(row=7,column=3) 72. def get\_selected\_row(self,event): 73. index=self.list1.curselection()[0] 74. self.selected\_tuple=self.list1.get(index) 75. self.e1.delete(0,END) 76. self.e1.insert(END,self.selected\_tuple[1]) 77. self.e2.delete(0,END) 78. self.e2.insert(END,self.selected\_tuple[2]) 79. self.e3.delete(0,END) 80. self.e3.insert(END,self.selected\_tuple[3]) 81. self.e4.delete(0,END) 82. self.e4.insert(END,self.selected\_tuple[4]) 84. def view\_command(self): 85. self.list1.delete(0,END) 86. for row in database.view(): 87. self.list1.insert(END,row) 89. def search\_command(self): 90. self.list1.delete(0,END) 91. for row in database.search(self.title\_text.get(),self.author\_text.get(),self.year\_text.get(),self.isbn\_text.get()): 92. self.list1.insert(END,row) 94. def add\_command(self): 95. database.insert(self.title\_text.get(),self.author\_text.get(),self.year\_text.get(),self.isbn\_text.get()) 96. self.list1.delete(0,END) 97. 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()  And below you will also find the backend.py script in OOP:  #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() | | | |