**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **28/5/2020** | **Name:** | **CHESMI B R** |
| **Course:** | **Logic design** | **USN:** | **4AL16EC100** |
| **Topic:** | **Boolean equations for digital circuits, combinational circuits, design of 7 segment decorders with common anode display.** | **Semester & Section:** | **8th sem A sec** |
| **Github Repository:** | **chesmibr** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  Boolean Algebra is an algebra, which deals with binary numbers & binary variables. Hence, it is also called as Binary Algebra or logical Algebra. A mathematician, named George Boole had developed this algebra in 1854. The variables used in this algebra are also called as Boolean variables. The range of voltages corresponding to Logic ‘High’ is represented with ‘1’ and the range of voltages corresponding to logic ‘Low’ is represented with ‘0’. **Postulates and Basic Laws of Boolean Algebra** In this section, let us discuss about the Boolean postulates and basic laws that are used in Boolean algebra. These are useful in minimizing Boolean functions. **Boolean Postulates** Consider the binary numbers 0 and 1, Boolean variable xx and its complement x′x′. Either the Boolean variable or complement of it is known as literal. The four possible logical OR operations among these literals and binary numbers are shown below.  x + 0 = x  x + 1 = 1  x + x = x  x + x’ = 1  Similarly, the four possible logical AND operations among those literals and binary numbers are shown below.  x.1 = x  x.0 = 0  x.x = x  x.x’ = 0  These are the simple Boolean postulates. We can verify these postulates easily, by substituting the Boolean variable with ‘0’ or ‘1’. **Basic Laws of Boolean Algebra** Following are the three basic laws of Boolean Algebra.   * Commutative law * Associative law * Distributive law  **Commutative Law** If any logical operation of two Boolean variables give the same result irrespective of the order of those two variables, then that logical operation is said to be Commutative. The logical OR & logical AND operations of two Boolean variables x & y are shown below  x + y = y + x  x.y = y.x  The symbol ‘+’ indicates logical OR operation. Similarly, the symbol ‘.’ indicates logical AND operation and it is optional to represent. Commutative law obeys for logical OR & logical AND operations. **Associative Law** If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be Associative. The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.  x + y+zy+z = x+yx+y + z  x.y.zy.z = x.yx.y.z  Associative law obeys for logical OR & logical AND operations. **Distributive Law** If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be Distributive. The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.  x.y+zy+z = x.y + x.z  x + y.zy.z = x+yx+y.x+zx+z  Distributive law obeys for logical OR and logical AND operations.  These are the Basic laws of Boolean algebra. We can verify these laws easily, by substituting the Boolean variables with ‘0’ or ‘1’. **Theorems of Boolean Algebra** The following two theorems are used in Boolean algebra.   * Duality theorem * DeMorgan’s theorem  **Duality Theorem** This theorem states that the dual of the Boolean function is obtained by interchanging the logical AND operator with logical OR operator and zeros with ones. For every Boolean function, there will be a corresponding Dual function. **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.  **COMBINATIONAL CIRCUIT:**  Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following −   * The output of combinational circuit at any instant of time, depends only on the levels present at input terminals. * The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit. * A combinational circuit can have an n number of inputs and m number of outputs.  **Common Anode 7-Segment Display**  * ​For common anode apply +5 volts to vcc pin in series to a 510 ohm-1k ohm resistor. This resistor is very important always include it other wise your seven segment display will be damaged by over current. Note both the vcc pins are short so apply +5 volts on only one pin and leave other empty. * Ground the dp(decimal/display point) pin if you want it to illuminate for ever. If you to control dp(decimal/display point) led than connect it to some control system, microcontroller etc. * In common Anode the Cathode(-) side of led’s are connected to a,b,c,d,e,f,g pins of seven segment display.​ ​ * In common anode seven segment display’s led becomes lit when we ground any a,b,c,d,e,f,g pin. * Common Anode seven segment display’s color is usually gray. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:28/5/2020** |  | **Name: Chesmi** |  | |
| **Course: Python** |  | **USN:4AL16EC100** |  | |
| **Topic: Application 5: Build a desktop database application** |  | **Sem :8th sem**  **Section: A sec** |  | |
| **AFTERNOON SESSION DETAILS** | | | |
| **SESSION IMAGE :**        **REPORT:**  **CONNECT BACKEND TO FRONTEND**  Python has been the most trending programming language used for object oriented programming. With python you can run simple statement over and over again without having to compile a whole program of which it's output functionality is superb.  Of course, Python is an interactive programming which has a diverse range of options for GUI (Graphical User Interface) framework (help developers create GUI applications in an easy and secure manner).  There are exactly two ways to perform what you are looking for, i.e., to insert data from data services of your backend into your frontend layout as content.   1. Server Side Rendering- This is how most and traditional websites specifically those are based on some CMS technology uses. The idea is to have some HTML template files with some placeholder for content that are later pushed by the backend program before sending to client browsers. In Python, you can have Jinja template framework at your disposal. You can wrap up variables into double braces and later providing values for each variable and compile them to static HTML file to render them on browser. 2. Client Side Rendering (Ajax)- This is the approach taken by progressive web applications. You will most probably use some JavaScript framework for the View and use some of its data services or another third party library to fetch data from Backend API endpoint and then update the view accordingly. Some of popular frameworks are- React, Vue (VDOM) or Angular, Mithril etc. 3. Hybrid- You may also however combine these two technique by using some server side rendering with a JavaScript plugin library with Ajax support such as jQuery.   **USER INTERFACE DESIGN**  There are many graphical user interface (GUI) toolkits that you can use with the Python programming language. The big three are Tkinter, wxPython, and PyQt. Each of these toolkits will work with Windows, macOS, and Linux, with PyQt having the additional capability of working on mobile.  A graphical user interface is an application that has buttons, windows, and lots of other widgets that the user can use to interact with your application. A good example would be a web browser. It has buttons, tabs, and a main window where all the content loads.  Interfaces play an important role in software engineering. As an application grows, updates and changes to the code base become more difficult to manage. More often than not, you wind up having classes that look very similar but are unrelated, which can lead to some confusion. In this tutorial, you’ll see how you can use a **Python interface** to help determine what class you should use to tackle the current problem.  nterfaces in Python are handled differently than in most other languages, and they can vary in their design complexity. By the end of this tutorial, you’ll have a better understanding of some aspects of Python’s data model, as well as how interfaces in Python compare to those in languages like Java, C++, and Go.  At a high level, an interface acts as a **blueprint** for designing classes. Like classes, interfaces define methods. Unlike classes, these methods are abstract. An **abstract method** is one that the interface simply defines. It doesn’t implement the methods. This is done by classes, which then **implement** the interface and give concrete meaning to the interface’s abstract methods.  Python’s approach to interface design is somewhat different when compared to languages like [Java](https://realpython.com/oop-in-python-vs-java/), Go, and C++. These languages all have an interface keyword, while Python does not. Python further deviates from other languages in one other aspect. It doesn’t require the class that’s implementing the interface to define all of the interface’s abstract methods.   1. FIXING A BUG 2. def get\_selected\_row(event): 3. try: 4. global selected\_tuple 5. index=list1.curselection()[0] 6. selected\_tuple=list1.get(index) 7. e1.delete(0,END) 8. e1.insert(END,selected\_tuple[1]) 9. e2.delete(0,END) 10. e2.insert(END,selected\_tuple[2]) 11. e3.delete(0,END) 12. e3.insert(END,selected\_tuple[3]) 13. e4.delete(0,END) 14. e4.insert(END,selected\_tuple[4]) 15. except IndexError: 16. pass  Explanation The error was fixed by simply implementing a try and except block. When the get\_selected\_row  function is called, Python will execute the indented block under try . If there is an IndexError, none of the lines under try  will be executed; the line under except  will be executed, which is pass. The pass statement means "do nothing". Therefore the function will do nothing when there's an empty listbox. | | | |