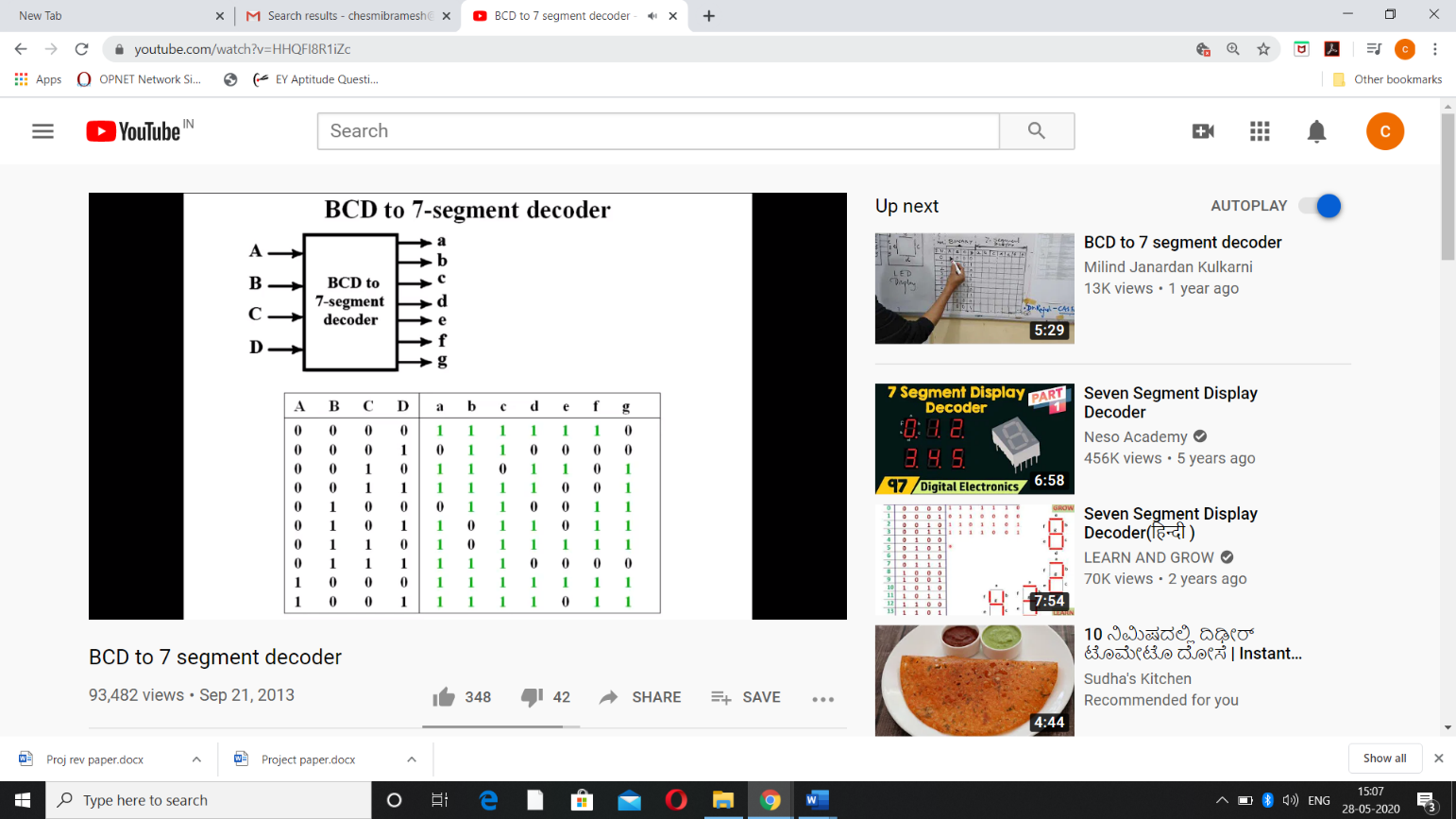
DAILY ASSESSMENT FORMAT

|  |  |  |  |
| --- | --- | --- | --- |
| Date | 28/05/2020 | Name: | Prajna |
| Course: | Logic design | USN: | 4AL16EC047 |
| Topic: | Boolean equations for digital circuits, combinational circuits, design of 7 segment decoders with common anode display. | Semester &  Section: | 8 “A” |
| FORENOON SESSION DETAILS | | | |

****

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.

|  |  |  |  |
| --- | --- | --- | --- |
| Date | 28/05/2020 | Name: | Prajna |
| Course: | PYTHON | USN: | 4AL16EC047 |
| Topic: | 1.Strings  2.Boolean  3.Operators  3.Collections | Semester &  Section: | 8 “A” |
| AFTERNOON SESSION DETAILS | | | |

## String Literals:

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print()function:

### Example

print("Hello")  
print('Hello')

## Assign String to a Variable:

Assigning a string to a variable is done with the variable name followed by an equal sign and the string.

### Example

a = "Hello"

## Multiline Strings:

You can assign a multiline string to a variable by using three quotes:

### Example

You can use three double quotes:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

Or three single quotes:

### Example

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

# Python Booleans

Booleans represent one of two values: True or False.

## Boolean Values

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer.

### Example

print(10 > 9)  
print(10 == 9)  
print(10 < 9)

When you run a condition in an if statement, Python returns True or False.

### Example

Print a message based on whether the condition is True or False:

a = 200  
b = 33  
  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

## Evaluate Values and Variables:

The bool() function allows you to evaluate any value, and give you True or False in return,

### Example

5542

print(bool("Hello"))  
print(bool(15))

### Example

Evaluate two variables:

x = "Hello"  
y = 15  
  
print(bool(x))  
print(bool(y))

## Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

### Example

The following will return True:

bool("abc")  
bool(123)  
bool(["apple", "cherry", "banana"])

## Some Values are False:

In fact, there are not many values that evaluates toFalse, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

### Example

The following will return False:

bool(False)  
bool(None)  
bool(0)  
bool("")  
bool(())  
bool([])  
bool({})

One more value, or object in this case, evaluates toFalse, and that is if you have an object that is made from a class with a \_\_len\_\_ function that returns 0 or False:

### Example

class myclass():  
  def \_\_len\_\_(self):  
    return 0  
  
myobj = myclass()  
print(bool(myobj))

## Functions can Return a Boolean

You can create functions that returns a Boolean Value:

### Example

Print the answer of a function:

def myFunction() :  
  return True  
  
print(myFunction())

You can execute code based on the Boolean answer of a function:

### Example

Print "YES!" if the function returns True, otherwise print "NO!":

def myFunction() :  
  return True  
  
if myFunction():  
  print("YES!")  
else:  
  print("NO!")

Python also has many built-in functions that returns a boolean value, like the isinstance()function, which can be used to determine if an object is of a certain data type:

### Example

Check if an object is an integer or not:

x = 200  
print(isinstance(x, int))

# Python Operators

## Python Operators:

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

## Python Arithmetic Operators:

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | x + y |
| - | Subtraction | x – y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation | x \*\* y |
| // | Floor division | x // y |

## Python Assignment Operators

Assignment operators are used to assign values to variables:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x – 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| //= | x //= 3 | x = x // 3 |
| \*\*= | x \*\*= 3 | x = x \*\* 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

Python Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## Python Logical Operators

Logical operators are used to combine conditional statements:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and | Returns True if both statements are true | x < 5 and  x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) |

## Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| is | Returns True if both variables are the same object | x is y |
| is not | Returns True if both variables are not the same object | x is not y |

## Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y |

## Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

Top of Form

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
* **Set** is a collection which is unordered and unindexed. No duplicate members.
* **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

## List :

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

### Example

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## Access Items

You access the list items by referring to the index number:

### Example

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[1])

### Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

### Example

Print the last item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])

### Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

### Example

Return the third, fourth, and fifth item:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])