# CS156 Module 3 Week 6 Programming Assignment #1

SUMMARY:

This programming exercise involves completing the code of one function named pl\_true that is part of Python module that contains functions to manipulate and interface with logic-based KBs. It is important to first complete the reading assignment for chapter 7 in the textbook, and review the lecture slides on Module 3 logic-based representation topic.

Is a function that takes as input a logical expression and a model. It returns True if the logical expression is true in the model and False/None otherwise. You will first need to review several functions in the logic.py, logic\_driver.py, and possibly other modules to understand how propositional statements are inserted, removed, and evaluated, and so forth in a KB and models.

The function in the logic.py module is partially completed. Within that function are several lines with the following text.

## <put the correct programming statement here.>

The inserted text indicates places where you need to insert Python statements/code to complete the implementation of the pl\_true function. The file named “correct\_pl\_true\_output.txt” contains the output one should get if the code for pl\_true is completed correctly. There are comments in the pl\_true function that are intended to provide hints and guidance about what your inserted code should accomplish. In each place where the

## <put the correct programming statement here.>

text appears, only one or two lines of code is needed … assuming you understand what needs to be done at each location in the code.

INSTRUCTIONS:

Consider the problem of deciding whether a propositional logic sentence is true in a given model. Write a recursive algorithm pl\_true(*s*,*m*) that returns *True* if and only if the sentence *s* is true in the model *m* (where *m* assigns a truth value for every symbol in *s*).

Completing this assignment will require you to read and understand sections 7.1 to and including 7.4 in the required textbook. After this programming assignment is completed, you will have good foundational knowledge and experience with propositional logic representation of knowledge. You will also begin to obtain a good sense of how the surrounding code works, how inferencing can be carried out, and so forth.

You will complete the assignment by modifying an extending the function named pl\_true that is located in the Python file named logic.py . The pl\_true function has a statement at the beginning that describes the overall purpose of the function. There are comments liberally placed in the function that described what the inserted code should accomplish. This is indicated by the text

## “<put the correct programming statement here.>”

Each place where the above text appears, you are expected to remove that comment and insert your code that accomplishes what the preceding comments indicate what needs to be accomplished.

Between reading the textbook, instruction provided in lecture, and reviewing the code, and your understanding of logic you should be able to complete this assignment.

If your code for pl\_true is correct, running the driver for the logic.py code should produce exactly the same output as contained in a file named correct\_pl\_true\_output.txt .

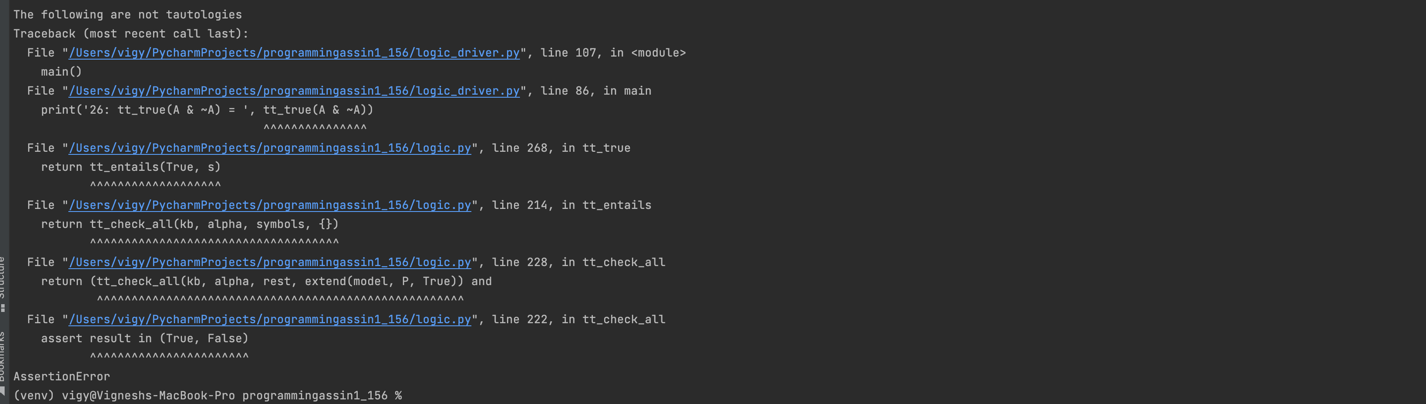
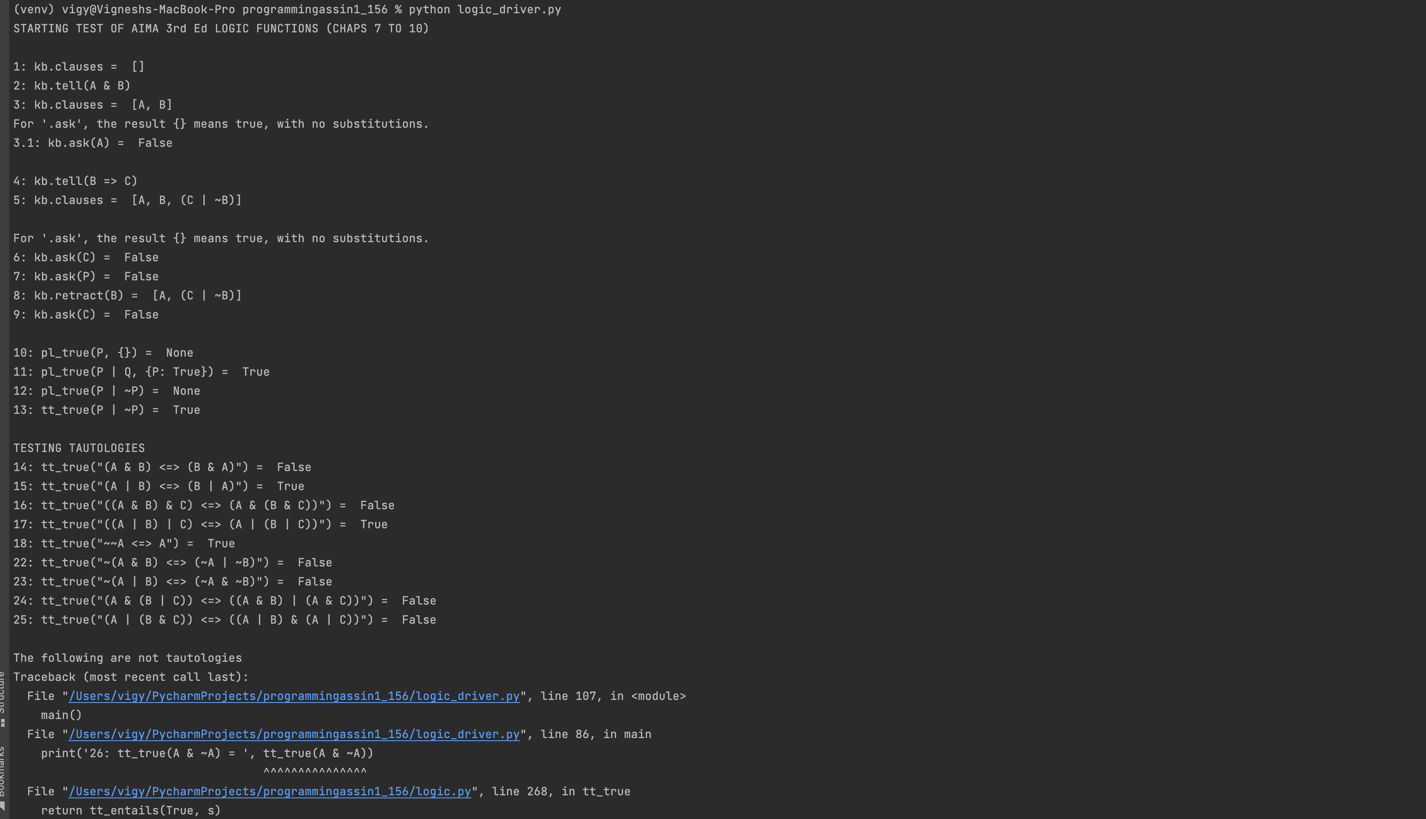
To begin working on this assignment, follow the steps below.

1. Download the “CS156\_Programming\_Assignment\_1.zip” file that contains all of the Python 3.9 compatible code and accompanying .txt files from the Canvas -> Module 3 -> Week 6 section on Canvas. The main driver module is named logic\_driver.py . The pl\_true function is located in the module named logic.py

1. Before making any modifications to any code, run the logic\_driver.py module and compare the output with that displayed in the correct\_pl\_true\_output.txt file. Note the differences. It is unlikely you will understand the difference unless you have read sections 7.1 to 7.4 in the textbook and attended the lectures on this topic.

1. To begin understanding how the code works, trace the execution of the code from the calls to various functions starting with the logic\_driver.py module. Some will call pl\_true while others will not. Match the execution thread you discover by tracing the execution with how proposition logic works, conversion of logical statements such as A => B to conjunctive normal form such as ~A or B, and so forth. Match what you see in the code with the discussion and your understanding form reading sections 7.1 to 7.4.

Answer: After executing code currently, found this result :



The functions in pl\_true are running the logical equivalance statememts as per symbols

Example: op == '~':  
 p = pl\_true(args[0], model)  
 if p is None:  
 return None  
 else:  
 return not p

~ is negation which gives the P as not P

Similarly this is or

p == '|':  
 result = False  
 for arg in args:  
 p = pl\_true(arg, model)  
 if p is True:  
 return True  
 if p is None:  
 result = None  
 return result

if anytime any P is true it returns true as it’s a or

Starting from statement :

1 Kb.clause means kb is empty

2 means we are adding A and B to kb

3 means kb now has A and B

3.1 means we are asking kb if A is there in it, which is true

4 we are telling kb that b=> c is a statement, which internally converts as ~b| c is true

5 now kb has a b and c| !b

6 we ask kb is C is there in it, as we have C | ~b is true and we know b is true so ~b is false and for c| ~b t be true , c has to be true. So by replacement of b we have ask c as true

7 we ask kb if P is there , which is false

8 retract from kb element b mans removing a clause from kb, so now kb has a ~b | c

9 we check if c is there , which is false, as there is no to derive c

10 we ask using propositional logic if P is true, as kb doesn’t have P or we cannot infer P from anything, so we give none as unsure of the result of P

11 we ask using propositional logic if P | Q is true, given P is true. As it’s a or statement if either clause P or Q is true the answer is true, hence the pl is true here

12 we ask using propositional logic if P | ~P is true, as we don’t know what P value is we cannot be sure of the result. So it returns None

13 truth table p | ~p means we can make a table using all possible values of P and ~p and see if this holds true for all cases

P. ~P. p|~p

T. F. T

T. T. T

F. T. T

T. T. T

In all cases the exp is true, hence tt is true

14 A & B ⬄ B & A means

((A & B) => (B & A)) & ((B & A) => (A & B)) by biconditional elimination

(~(A&B)| (B&A)) & (~(B & A)|(A&B)) by implication elimination

(~A|~B)| (B & A) & ((~B | ~A) | (A & B)) by deMorgan

From kb we have A

(~B | B) & (~B | B) by replacing A

By tt we have ~B | B as always true

T and T = true

15 A | B ⬄ B | A means

((A | B) => (B | A)) & ((B | A) => (A | B)) by biconditional elimination

(~(A|B)| (B|A)) & (~(B | A)|(A|B)) by implication elimination

(~A&~B)| (B | A) & ((~B & ~A) | (A | B)) by deMorgan

From kb we have A

(False | true) & (false | true) by replacing A

By tt we have true & true as always true

T and T = true

16 ((A&B)&C) ⬄ (A& (B&C)) means

By distributive law A& (B&C) = (A&B) & (A &C)

(A & (B&C)) = ((A&B) & (A&C))

BY ASSOCIATE LAW ((A&B)&C) (A& (B&C))

((A&B)&C) = (A& (B&C))

Using above two we have ((A&B)&C) ⬄ (A& (B&C)) true

17 ((A | B) | C) <=> (A | (B | C))") = True

Associative LAW (A | B)|C = A |(B|C)

((A|B)|C) = (A|(B|C))

A |(B|C) = (A|B)|C

(A|(B|C)) = ((A|B)|C)

Hence above is true

18 ~~A ⬄ A double negation is A ⬄ A

~A | A is always true

22 ~(A & B) <=> (~A | ~B)")

Using demorgan ~(A&B) means ~A | ~B which is same as the right side argument and

A⬄ A is always true, hence above is also true

23 ~(A | B) <=> (~A & ~B)

Using demorgan ~(A|B) means ~A & ~B which is same as the right side argument and

A⬄ A is always true, hence above is also true

24 (A & (B | C)) <=> ((A & B) | (A & C))

Using distributive rule

A& (B|C) = (A&B) | (A &C) which is same as right side and

A⬄ A is always true, hence above is also true

25 (A | (B & C)) <=> ((A | B) & (A | C))

Using distributive rule

A| (B&C) = (A|B) & (A |C) which is same as right side and

A⬄ A is always true, hence above is also true

26 A&~A

A ~A A&~A

T. F. F

F. T. F

Always false

27 (A&B) , A is true as per kb but e don’t know B so tt is false

33 x, x unify as same vaue so nothing added to model

34 x , 3 x is 3 is added to model

35 to\_cnf(p&Q) | (~P & ~Q))

Distributive of P

(p|~P) & (p|~Q) & (Q|~P) & (Q|~Q)

Got the cnf modification

1. Then write down, in pseudocode, what you need to do to modify pl\_true. Then incrementally make modifications to the function and run it to check its output with that displayed in correct\_pl\_true\_output.txt . Modify, correct, update, and extend as needed.

Answer:

(a)op == '&':  
 result = True  
 for arg in args:  
 return arg #<put the correct programming statement here.> Replace return arg with correct statement  
 if p is False:  
 return False  
 if p is None:  
 result = None  
 return result  
#

This is not updated as per need.

Hence, corrected value is

Pseudocode for &

For arg in args:

If (arg != true)

P= false

Else P=true.

Every single clause or symbol should be true for this & to return true

Modified code:

op == '&':

result = True

for arg in args:

p = pl\_true(arg, model)

if p is False:

return False

if p is None:

result = None

return result

(b) if op == '==>':  
 return exp #<put the correct programming statement here.> Replace return exp with correct statement

=> is not properly written

=> means not a or b

Pseudocode for =>

If op ‘=>’ pl\_true(~p | q)

Updatec ode:

if op == '==>':

return pl\_true(~p | q, model)

(c) elif op == '<==':  
 return exp #<put the correct programming statement here.> Replace return exp with correct statement

<= is also not created properly , this should be

<= means p|~q

Pseudocode for <=

If op= ‘<=’ pl\_true(p|~q)

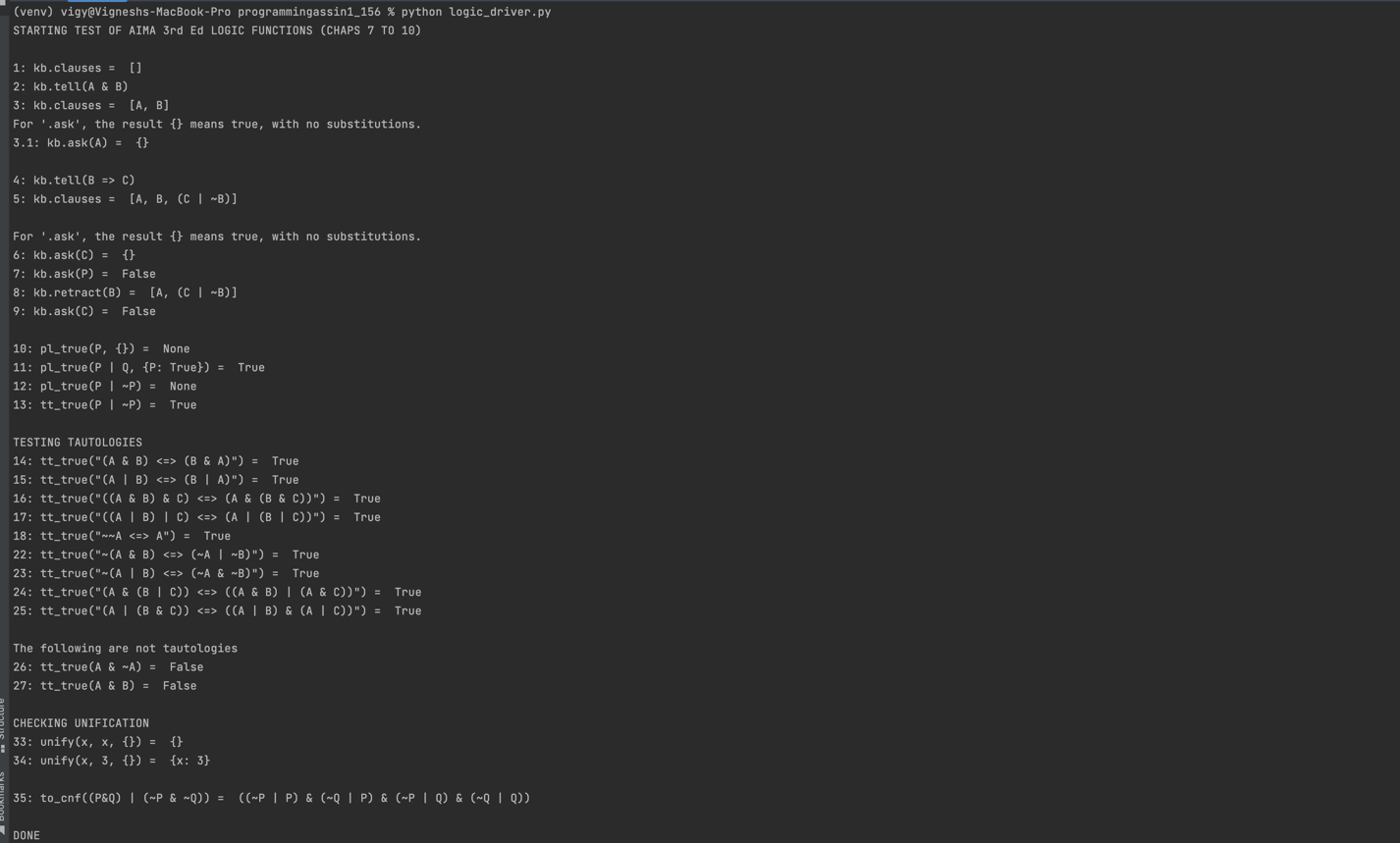
Updated code

op == '<==':

return pl\_true(p | ~q, model)

After running the code with the above changes, getting the below result :

Which is same as the output for pl\_true file.

FYI: there is just a small prob in the output file that is present in the assignment, the kb.ask(A) should return true as A is already is kb. But as per the tx file its marked as false. I had a discussion with the professor and he mentioned it should be true. So, the corrected output is as below which is exactly.

Do not spin your wheels too long if and when you are uncertain. The instructor can provide hints or suggestions as appropriate to help you figure out the answer on your own.

Best of luck.