Illustration:

1) Representing Propositional Sentence
   The sentences in propositional logic is represented using nested lists which is Python-style.
   For example:

$$(A \wedge B) \vee C$$

   should be written as:

["or", ["and", "A", "B"], "C"]

   There are only seven connectives in the coding:
   - "not" negation
   - "and" conjunction
   - "or" disjunction
   - "implies" implication
   - "iff" biconditional implication
   - "xor" the exclusive or
   - "ifte" the if-then-else operator

   Where the following assumptions is adopted:
   - The order of the literals does not matter. For example, ["and", "A", "B"] equals to ["and", "B", "A"]
   - Assume that the inputs are always logically correct, i.e. "not" always has 1 parameter; "implies", "iff", "excl" always have **2** parameters, "ifte" always has **3** parameters, "and" and "or" have **2 or more** parameters.

2) The converting process:
   The coding is built and modified based on an existing implementation [1].
   The programming follows the following process:

Step 1	Remove the exclusive-or operator
Step 2	Remove the IFTE operator
Step 3	Remove the biconditional operator
Step 4	Remove the implication operator
Step 5	Eliminate the double negation and implement De Morgan Law
Step 6	Remove the mighty redundant brackets generated (e.g. [["A"]])
Step 7	Apply the Distributivity Law
Step 8	Remove the duplicated symbols (e.g. ["and", "A", "A"])
Step 9	Remove the duplicated operators (e.g. ["and", "A", ["and", "B", "C"]])
Step 10	Double check the Distributivity Law and remove duplicated symbols and operators

3) Instructions:
   1. The code is written in Python version 2.7. It is ideally to run the code under a Python GUL which is compatible with Python version 2.7, for example **PyScripter for Python 2.7** (This tool exits in the University of Aberdeen IT toolkit.)

2. Several tested propositional formulas are given at the end of the programming. By deleting the symbol "#" in the very front, the user is able to test the specific propositional formula.

```
""" TEST PROPOSITIONAL FORMULA """
#end=convert_to_cnf('["ifte","A","B","C"]')       #test for ifte function
#end=convert_to_cnf('["xor","B","C"]')            #test for xor function
```

However, one could also input the customized testing sentence by following the format as:

$$end = convert\_to\_cnf('\boldsymbol{your\ customized\ input}')$$

Where ***your customized input*** should follow the Python_style rules as illustrated in the previous section of **Representing Propositional Sentence.**

3. By running the Python GUL, the converted CNF will display in the Python Interpreter window as shown below:

```
*** Remote Interpreter Reinitialized  ***
>>>
The chosen test is:  ["and",["or","A","A"],["or","A","A"],"A","A"]
The convered CNF is:  ['A']
>>>
```

4) Performance:
The outcome is satisfied for most of the situation. Extreme cases are considered for example ["not", ["not",["and", "A", "A"]]] which will generated [["A"]], therefore a function **remove_brackets(exp)** was developed to modify the ultimate outcome which would modify [["A"]] to ["A"].
For ifte operators, due to the complexity of the formula expansion, the ultimate outcome generated by the code could not be justified.

5) Bug exists
Bug exists in the distributivity module which was taken from the reference. As shown in the source code below, the distributivity law only covers the first three element in a list. Outranged element will not be calculated, for example:
For formula **["or","A", ["and","B","C","D"], ["and","E","F","G"]]**, the last element **["and","E","F","G"]** will be ignored.

```
#The following function is <Taken from Reference>
#Function to implement distributive law
def distribution_or_over_and(exp):
    if(exp[0]=="or"):
        if(len(exp[2])>1  and exp[2][0]=="and"):
            temp1=exp[1]
            temp2=exp[2]
            del exp[:]
            exp.append("and")
            for item1 in temp2:
                if item1=="and":
                    continue
                exp.append(["or",temp1,item1])
        elif(len(exp[1])>1  and exp[1][0]=="and"):
            temp1=exp[1]
            temp2=exp[2]
            del exp[:]
            exp.append("and")
            for item2 in temp1:
                if item2=="and":
                    continue
                exp.append(["or",item2,temp2])

    for item in exp:
        if(len(item)>1 ):
            distribution_or_over_and(item)

    return exp
```

Restricted by my programming skill and Python coding knowledge, an optimized distributivity could not be developed.

References

[1] Vivek, "Github," 5 May 2015 . [Online]. Available: https://github.com/Vivek-100/CNF_Converter. [Accessed 12 10 2017].