Lab 8 – Boolean Expression Evaluator

In this exercise you will be creating a class that can evaluate Boolean expressions written in both RPN and in Infix notation.

We will use assume that the input statements are well formatted (no unexpected characters) and spaces between each item in the string. Feel free to write the code to format the input string if you want.

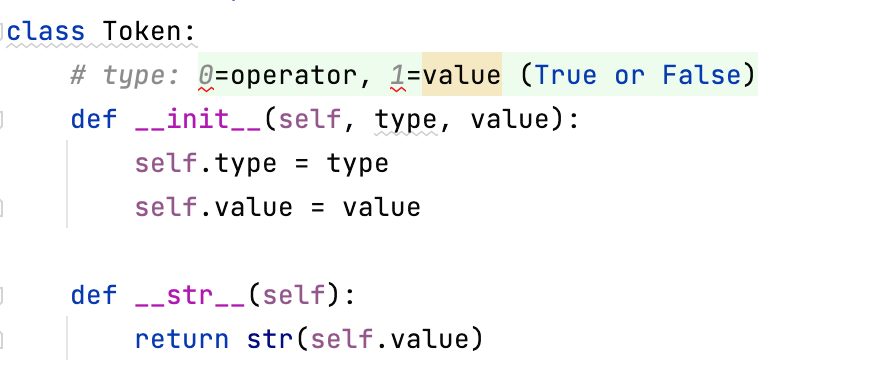
An RPN Boolean expression is an expression of the form: “T F ^” or “T T F v ^”. We will use the letter “^” to represent the operator AND. We will use the letter “v” to represent the operator OR.

An infix Boolean expression is an expression of the form “T ^ F” or “T ^ ( T v F )”.

# Part 1 – Evaluation of RPN Expressions

Evaluating an RPN expression is relatively simple and description of how to evaluate these postfix notation expressions is easy to find on the internet. Well, it should be easy to find one that can evaluate integers but of course we want to evaluate Boolean expressions. Even if you could find one online, I would still suggest that you write this one from scratch.

Create the following class to hold the tokens which will be either operators or values:



Create a class called BooleanEvaluator with the following functions:

* load\_rpn(str): Load an RPN expression into the class. I would recommend that you create a list of Token objects based on the input string. I would suggest converting T and F into actual True and False values.
* display\_rpn(): This will print the RPN expression that was loaded. It should look like the string that was just loaded. We will need this for debugging and for Part 2 of this exercise.
* evaluate(): This will calculate the result of the RPN expression currently stored and return either true or false.

# Part 2 – Infix to Postfix Conversion

Add a function called load\_infix(str) that will convert a string in infix notation and convert it to postfix format. Once it has been converted into postfix notation inside the ADT, you would be able to evaluate it:

For example the following code:

be = BooleanEvaluator()  
be.load\_infix("T ^ ( T v F )")  
be.display\_rpn()  
print("Result:", be.evaluate())

should produce the output:

True True False v ^

Result: True

No “official” unit test is required but please provide at least 4 statements involving ^ and v operators with parentheses. For each test case print the RPN equivalent and the evaluated expression.