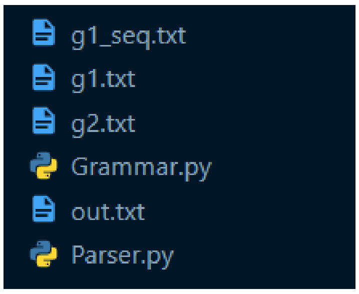
Code review for https://github.com/912-Cireap-Bogdan/LFTC/tree/main/Lab5-Parser

LANGUAGE CHOICE
Python is well suited to this laboratory, being easy to read/write, but deep enough to handle moderately complex code easily.

FILE STRUCTURE
File structure is great. The Grammar and the Parser are in separate files, each of them containing one class and its corresponding wrapper functions for testing. Input/output files could have been placed in different folders though.

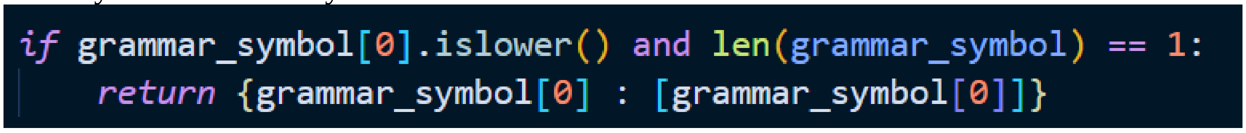


CODE AESTHETICS / STYLE
Spacing is good, code blocks are clearly separated by function. Naming is solid, on the lengthy side, but autocomplete makes writing long names a non-issue. Better to be too explicit than too vague. snake_case is used, as tradition entails. In certain places there are too many spaces between lines for no good reason.

DOCUMENTATION
Healthy amount of comments, with very good explanations for certain more complicated parts of the code. A couple of functions are missing any comments, though.

GRAMMAR CLASS
Parses the grammar from file and formats it into a usable structure in memory. Code is concise and clean. Not much to add, as there isn't much code to begin with.

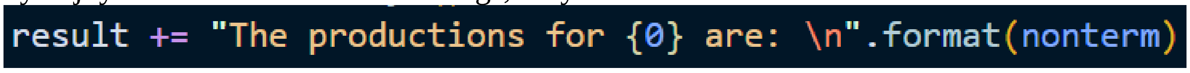
PARSER CLASS
This is supposed to parse a given sequence according to the grammar, using the LL1 strategy. The First and Follow functions are implemented, parts of the parsing strategy. They are well explained, and the code is explicit. A lot of smart tools are used to shorten the code, such as this dictionary created on the fly in the First function:



```
if grammar_symbol[0].islower() and len(grammar_symbol) == 1:
    return {grammar_symbol[0] : [grammar_symbol[0]]}
```

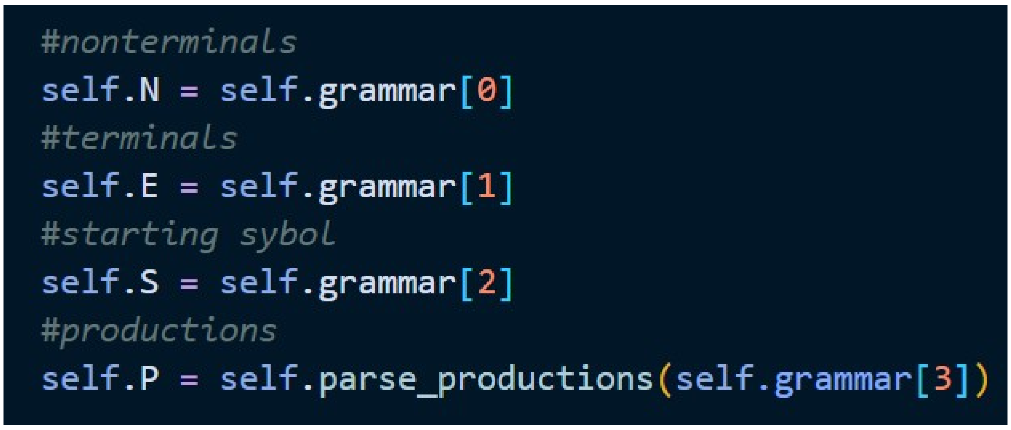
GENERAL COMMENTS

Really enjoyed the use of formatted strings, very clean



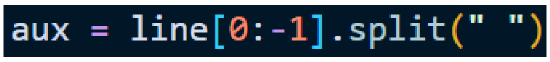
```
result += "The productions for {0} are: \n".format(nonterm)
```

Would have preferred more explicit names instead of comments here. 'self.grammar' being an array is excusable by the fact that Python doesn't allow quick creation of objects/models, but it is an unwieldy way to code it nonetheless.



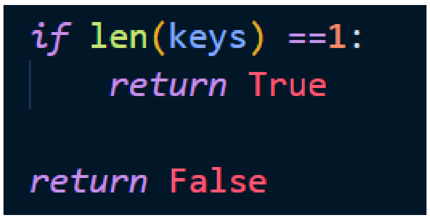
```
#nonterminals
self.N = self.grammar[0]
#terminals
self.E = self.grammar[1]
#starting sybol
self.S = self.grammar[2]
#productions
self.P = self.parse_productions(self.grammar[3])
```

Smart use of indexes here



```
aux = line[0:-1].split(" ")
```

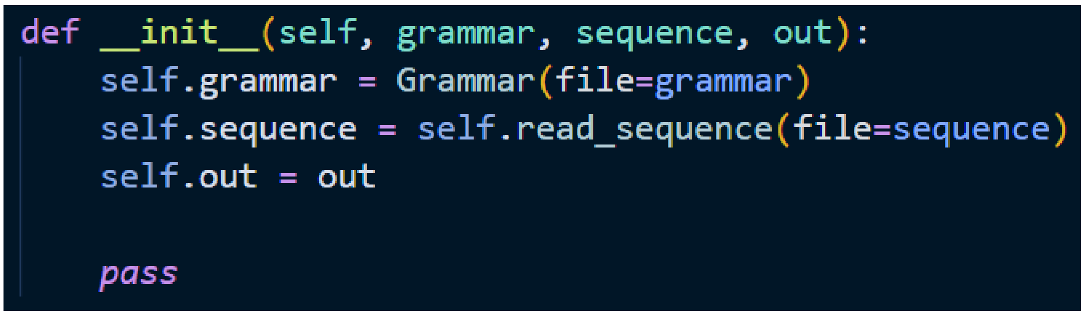
Could have used a shortened if here no problem.



```
if len(keys) ==1:
    return True

return False
```

Not sure what this pass is doing here in the init function. Maybe we’re missing some python thing?



```
def __init__(self, grammar, sequence, out):
    self.grammar = Grammar(file=grammar)
    self.sequence = self.read_sequence(file=sequence)
    self.out = out

    pass
```