Please make this submission as the current directory. On terminal, type the command:

python parser.py atis-grammar-cnf.cfg show the flights .

Note: I was unable to debug my program but I have implemented a recognizer, a parser and a parse tree function in my code.
However, since the code was not getting debugged, I was not able to receive the required output.
I will try this question again during the Christmas break.

```
#Assignment number 3
import sys
import os.path

def parser(grammar_filename, sentence):
        """
        filename where grammar is recorded,sentence to be parsed.
        """
        grammar = getGrammar(grammar_filename)

        nodes_back = cky(grammar, sentence.split())

        printParseTrees(nodes_back)

def cky(grammar, sentence):
        """
        cky() takes sentence and parses it according to the provided grammar.
        """
        n = len(sentence)
        table = [[[] for i in range(n + 1)] for j in range(n + 1)]
        nodes_back = [[[] for i in range(n + 1)] for j in range(n + 1)]

        for j in range(1, n + 1):
                for rule in grammar:
                        if [sentence[j - 1]] in grammar[rule]:
                                table[j - 1][j].append(rule)
                                nodes_back[j - 1][j].append(
                                        Node(rule, None, None, sentence[j - 1]))

                # Loop over diagonally in the table and fill in the fields using
                # the rules of the grammar. I check subnodes to find out whether
                # a rule applies or not.
                for i in reversed(range(0, j - 1)): #(j - 2, 1) goes to 0
                        for k in range(i + 1, j): # goes to j - 1
                                for rule in grammar:
                                        for derivation in grammar[rule]:
                                                if len(derivation) == 2:
                                                        B = derivation[0]
                                                        C = derivation[1]

                                                        if B in table[i][k] and C in table[k][j]:
                                                                table[i][j].append(rule)

                                                                for b in nodes_back[i][k]:
                                                                        for c in nodes_back[k][j]
:
                                                                                if b.root == B an
d \
                                                                                        c.root == C:
                                                                                                nodes_bac
k[i][j].append(
                                                                                                        N
ode(rule, b, c, None))

        return nodes_back[0][n]

def printParseTrees(nodes_back):
        """
        printParseTrees() takes a list of root nodes and prints the ones that
        start with a 'SIGMA' - START Symbol.
        """
        check = False
        for node in nodes_back:
                if node.root == 'SIGMA':
                        print(getParseTree(node, 3))
                        print()
                        check = True

        if not check:
                print('The given sentence is not valid according to the grammar.')

def getParseTree(root, indent):
```

```
        """
        getParseTree() takes a root and constructs the tree in the form of a
        string. Right and left subtrees are indented equally, providing for
        a nice display.
    """
        if root.status:
                return '(' + root.root + ' ' + root.terminal + ')'

        # Calculates the new indent factors that we need to pass forward.
        new1 = indent + 2 + len(root.left.root) #len(tree[1][0])
        new2 = indent + 2 + len(root.right.root) #len(tree[2][0])
        left = getParseTree(root.left, new1)
        right = getParseTree(root.right, new2)
        return '(' + root.root + ' ' + left + '\n' \
                        + ' '*indent + right + ')'

def getGrammar(grammar_filename):
        """
        getGrammar() takes the filename of the file where our grammar rules are
        listed and reads these rules into a dictionary. The dictionary with the
        rules recorded is returned.

        Rules:
        - All lines are of the form X --> Y Z, X --> Y, X --> t.
        - Strings beginning with an uppercase letter are nonterminals.
        - Strings beginning with a lowercase letter are terminals.

        """
        try:
                grammar_text = open(sys.argv[1], 'r')
        except:
                printError(1)

        grammar = {}
        # Loops over each line in the grammar file given to record the
        # grammar rules.
        for line in grammar_text:
                # We do not want to read the comments.
                if line[0] != '#':
                        # Finds the different parts of the rule.
                        rule = line.split(' -> ')

                        rule[0] = rule[0].strip()
                        rule[1] = rule[1].strip()

                        # Right hand side needs to contain one or two elements.
                        # If two elements: neither can start with lower letter.
                        # If one element: can start with lower letter.
                        right_side = rule[1].split()
                        if (len(right_side) > 2) or (len(right_side) == 0):
                                printError(1)

                        elif len(right_side) == 2:

                                if right_side[0][0] == right_side[0][0].lower():
                                        printError(1)

                                elif right_side[1][0] == right_side[1][0].lower():
                                        printError(1)

                        # Left hand side can only contain one element and that element
                        # needs to be uppercase for the first letter.
                        left_side = rule[0].split()
                        if len(left_side) != 1:
                                printError(1)

                        elif left_side[0][0] != left_side[0][0].upper():
                                printError(1)

                        # If we have seen a derivation before, we add it to the list.
                        if rule[0] in grammar:
                                if right_side in grammar[rule[0]]:
```

```
                                printError(1)
                        else:
                                grammar[rule[0]].append(right_side)
                # If we have not seen a derivation before we need to add it to
                # the dictionary.
                else:
                        grammar[rule[0]] = [right_side]

        return grammar

def printError(num):
        """
        printError() prints out an error message and exits the program.

        """
        if num == 1:
                print('Error in the grammar file provided.')
        else:
                print('Error.')


def __init__(self, root, left, right, end):
                """
                Constructor for the Node class. Root, left, right, terminal and status
                are set up here. Status is infered from whether a terminal value is
                provided or not.
                """
                self._root = root
                self._left = left
                self._right = right
                self._terminal = end
                self._status = True
                if end == None:
                        self._status = False

def root(self):
        """
        root allows to get the root of the node.
        """
        return self._root

def left(self):
        """
        left allows to get the left subtree of the node.
        """
        return self._left

def right(self):
        """
        right allows  to get the right subtree of the node.
        """
        return self._right

def status(self):
        """
        status allowsto get the status of the node.
        """
        return self._status

def terminal(self):
        """
        terminal allows to get the terminal value of the node.
        """
        return self._terminal

def main():
        if len(sys.argv) != 3:
            printError(0)
        elif not os.path.isfile(sys.argv[1]):
            printError(0)
        parser(sys.argv[1], sys.argv[1])
```

```
if __name__ == '__main__':
        main()
```