**11.      Implement a simple top-down parser for context-free grammars using python.**

**Program:**

class Parser:

def \_\_init\_\_(self, input\_string):

self.tokens = input\_string.split()

self.current\_token = None

self.index = 0

def get\_next\_token(self):

if self.index < len(self.tokens):

self.current\_token = self.tokens[self.index]

self.index += 1

else:

self.current\_token = None

def parse(self):

self.get\_next\_token()

self.expression()

def expression(self):

self.term()

while self.current\_token in ['+', '-']:

operator = self.current\_token

self.get\_next\_token()

self.term()

print(f' {operator} ', end='')

def term(self):

self.factor()

while self.current\_token in ['\*', '/']:

operator = self.current\_token

self.get\_next\_token()

self.factor()

print(f' {operator} ', end='')

def factor(self):

if self.current\_token.isdigit():

print(self.current\_token, end='')

self.get\_next\_token()

elif self.current\_token == '(':

self.get\_next\_token()

self.expression()

if self.current\_token == ')':

self.get\_next\_token()

else:

raise SyntaxError("Expected ')'")

else:

raise SyntaxError("Invalid token")

input\_string = "3 + 4 \* ( 2 - 1 )"

parser = Parser(input\_string)

parser.parse()

**12.      Implement an Earley parser for context-free grammars using a simple python program.**

**Program:**

class State:

def \_\_init\_\_(self, rule, dot\_index, start\_index):

self.rule = rule

self.dot\_index = dot\_index

self.start\_index = start\_index

def \_\_eq\_\_(self, other):

return self.rule == other.rule and self.dot\_index == other.dot\_index and self.start\_index == other.start\_index

def \_\_hash\_\_(self):

return hash((self.rule, self.dot\_index, self.start\_index))

def \_\_str\_\_(self):

return f'{self.rule} : {"".join(self.rule)} - {self.dot\_index} - {self.start\_index}'

def earley\_parse(grammar, sentence):

chart = [[] for \_ in range(len(sentence) + 1)]

start\_rule = next(iter(grammar))

start\_state = State(start\_rule, 0, 0)

chart[0].append(start\_state)

for i in range(len(sentence) + 1):

while True:

added = False

for state in chart[i]:

if state.dot\_index < len(state.rule) and state.rule[state.dot\_index] in grammar:

non\_terminal = state.rule[state.dot\_index]

for rule in grammar[non\_terminal]:

new\_state = State(rule, 0, i)

if new\_state not in chart[i]:

chart[i].append(new\_state)

added = True

elif state.dot\_index < len(state.rule) and state.rule[state.dot\_index] == sentence[i - 1]:

new\_state = State(state.rule, state.dot\_index + 1, state.start\_index)

if new\_state not in chart[i]:

chart[i].append(new\_state)

added = True

elif state.dot\_index == len(state.rule):

for s in chart[state.start\_index]:

if s.dot\_index < len(s.rule) and s.rule[s.dot\_index] == state.rule[0]:

new\_state = State(s.rule, s.dot\_index + 1, s.start\_index)

if new\_state not in chart[i]:

chart[i].append(new\_state)

added = True

if not added:

break

for i in range(len(chart)):

print(f"Chart[{i}]:")

for state in chart[i]:

print(state)

grammar = {

'S': [['NP', 'VP']],

'NP': [['DET', 'N'], ['N']],

'VP': [['V', 'NP']],

'DET': ['the', 'a'],

'N': ['man', 'ball', 'woman', 'table'],

'V': ['hit', 'took', 'saw', 'liked']

}

sentence = ['the', 'man', 'hit', 'the', 'table']

earley\_parse(grammar, sentence)

**13.      Generate a parse tree for a given sentence using a context-free grammar using python program.**

**Program:**

import nltk

def generate\_parse\_tree(sentence, grammar):

parser = nltk.ChartParser(grammar)

tokens = nltk.word\_tokenize(sentence)

parse\_trees = list(parser.parse(tokens))

return parse\_trees

simple\_grammar = nltk.CFG.fromstring("""

S -> NP VP

NP -> Det N

VP -> V NP

Det -> 'the' | 'a'

N -> 'dog' | 'cat'

V -> 'chased' | 'caught'

""")

sentence = "the dog chased a cat"

parse\_trees = generate\_parse\_tree(sentence, simple\_grammar)

for i, tree in enumerate(parse\_trees):

print(f"Parse Tree {i + 1}:")

tree.pretty\_print()

print()

**14.      Create a program in python to check for agreement in sentences based on a context-free grammar's rules.**

**Program:**

import nltk

def check\_agreement(sentence, grammar):

parser = nltk.ChartParser(grammar)

tokens = nltk.word\_tokenize(sentence)

try:

parse\_tree = next(parser.parse(tokens))

return True

except StopIteration:

return False

agreement\_grammar = nltk.CFG.fromstring("""

S -> NP\_SG VP\_SG | NP\_PL VP\_PL

NP\_SG -> Det\_SG N\_SG

NP\_PL -> Det\_PL N\_PL

VP\_SG -> V\_SG

VP\_PL -> V\_PL

Det\_SG -> 'the'

Det\_PL -> 'the'

N\_SG -> 'cat'

N\_PL -> 'cats'

V\_SG -> 'chases'

V\_PL -> 'chase'

""")

sentence1 = "the cat chases"

sentence2 = "the cats chases"

result1 = check\_agreement(sentence1, agreement\_grammar)

result2 = check\_agreement(sentence2, agreement\_grammar)

print(f"Sentence 1 Agreement: {'Yes' if result1 else 'No'}")

print(f"Sentence 2 Agreement: {'Yes' if result2 else 'No'}")

**15.      Implement probabilistic context-free grammar parsing for a sentence using python.**

**Program:**

import nltk

pcfg\_grammar = nltk.PCFG.fromstring("""

S -> NP VP [1.0]

NP -> Det N [0.5] | N [0.3] | N PP [0.2]

VP -> V NP [0.9] | VP PP [0.1]

Det -> 'the' [0.8] | 'a' [0.2]

N -> 'dog' [0.4] | 'cat' [0.3] | 'park' [0.3]

V -> 'chased' [0.7] | 'caught' [0.3]

PP -> P NP [1.0]

P -> 'in' [0.6] | 'on' [0.4]

""")

pcfg\_parser = nltk.ViterbiParser(pcfg\_grammar)

sentence = "the dog chased the cat in the park"

tokens = nltk.word\_tokenize(sentence)

for tree in pcfg\_parser.parse(tokens):

tree.pretty\_print()

print("Probability:", tree.prob())