class State(enumerate):  
 NORMAL = 'q'  
 ERROR = 'e'  
 BACK = 'b'  
 FINAL = 'f'  
  
  
class Configuration:  
 def \_\_init\_\_(self, starting\_symbol):  
 self.state = State.NORMAL  
 self.index = 0  
 # array containing productions (ex: ('S', ['0', 'B']) or terminals  
 self.work\_stack = []  
 # array containing symbols (terminals or non-terminals)  
 self.input\_stack = [starting\_symbol]  
  
  
def recursive\_descendant(grammar, sequence):  
 config = Configuration(grammar.S)  
 while config.state != State.FINAL and config.state != State.ERROR:  
 if config.state == State.NORMAL:  
 if len(config.input\_stack) == 0 and config.index == len(sequence):  
 success(config)  
 elif len(config.input\_stack) == 0:  
 momentary\_insuccess(config)  
 else:  
 if config.input\_stack[0] in grammar.getNonTerminals():  
 expand(config, grammar)  
 else:  
 if config.index == len(sequence):  
 momentary\_insuccess(config)  
 elif config.input\_stack[0] == 'E':  
 config.work\_stack.append('E')  
 config.input\_stack = config.input\_stack[1:]  
 elif config.input\_stack[0] == sequence[config.index]:  
 advance(config)  
 else:  
 momentary\_insuccess(config)  
 else:  
 if config.state == State.BACK:  
 if config.work\_stack[-1] in grammar.E:  
 if config.work\_stack[-1] == 'E':  
 config.work\_stack.pop(-1)  
 else:  
 back(config)  
 else:  
 another\_try(config, grammar)  
  
 prod\_rules = []  
 if config.state == State.ERROR:  
 return False, []  
 else:  
 for prod in config.work\_stack:  
 if len(prod) > 1:  
 if prod[0] in grammar.P.keys():  
 if prod[1] in grammar.P[prod[0]]:  
 prod\_rules.append(prod)  
  
 return True, prod\_rules  
  
  
def expand(config, grammar):  
 # head of input stack is a non-terminal  
 non\_term = config.input\_stack[0]  
 first\_prod\_rhs = grammar.getProductionsFor(non\_term)[0] # array of symbols  
 config.work\_stack.append((non\_term, first\_prod\_rhs))  
 # remove first elem from input stack and replace it with its production  
 config.input\_stack = first\_prod\_rhs + config.input\_stack[1:]  
  
  
def advance(config):  
 # head of input stack is a terminal = current symbol from input  
 config.index += 1  
 config.work\_stack.append(config.input\_stack[0])  
 config.input\_stack = config.input\_stack[1:]  
  
  
def momentary\_insuccess(config):  
 # head of input stack is a terminal ≠ current symbol from input  
 config.state = State.BACK  
  
  
def back(config):  
 # head of working stack is a terminal  
 config.index -= 1  
 terminal = config.work\_stack.pop(-1)  
 config.input\_stack = [terminal] + config.input\_stack  
  
  
def another\_try(config, grammar):  
 # head of working stack is a non-terminal  
 (lhs, rhs) = config.work\_stack[-1]  
 productions = [production for production in grammar.getProductionsFor(lhs)]  
 next\_prod = get\_next\_production(rhs, productions)  
 if next\_prod:  
 config.state = State.NORMAL  
 config.work\_stack.pop(-1)  
 config.work\_stack.append((lhs, next\_prod))  
 config.input\_stack = config.input\_stack[len(rhs):]  
 config.input\_stack = next\_prod + config.input\_stack  
 elif config.index == 0 and lhs == grammar.S:  
 config.state = State.ERROR  
 else:  
 config.work\_stack.pop(-1)  
 if rhs == ['E']:  
 config.input\_stack = [lhs] + config.input\_stack  
 else:  
 config.input\_stack = [lhs] + config.input\_stack[len(rhs):]  
  
  
def success(config):  
 config.state = State.FINAL  
  
  
# helper function  
def get\_next\_production(prod, prods):  
 for i in range(len(prods)):  
 if prod == prods[i] and i < len(prods) - 1:  
 return prods[i + 1]  
 return None