**PRACTICAL -4**

**Name: Amarsingh Kashyap**

**Branch: CSE 2nd shift**

**Roll No: 1**

**Topic:** Parsing

**Platform:** Windows or Linux

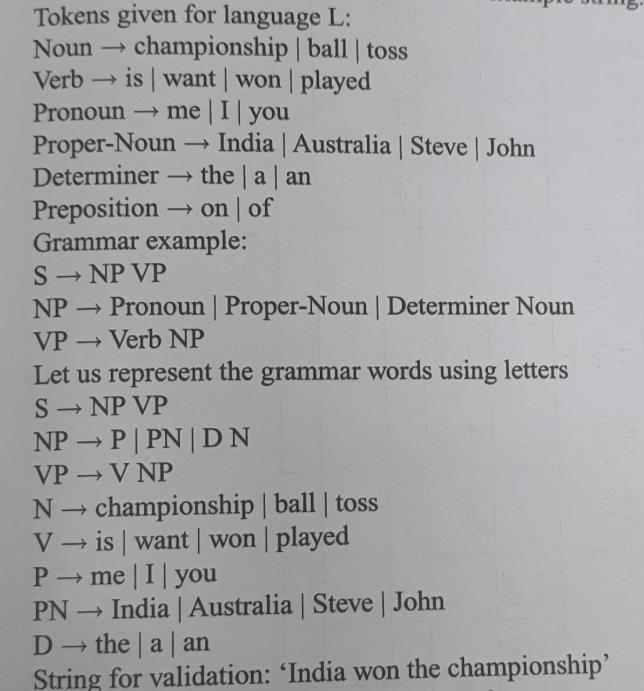
**Language to be used:** Python or Java (based on the companies targeted for placement)

# Aim:

1. *Write a program to validate a natural language sentence. Design a natural language grammar, compute and input the LL(1) table. Validate if the given sentence is valid based on the grammar or not.*

Input: NLP grammar and LL(1) parsing table (from file) Implementation: String parsing rules

Output: Each step in string parsing and whether the input string is valid or invalid.

**Sample NLP Grammar to validate a sentence:**

**CODE:**

**from** collections **import** deque  
buffer = deque()  
stack = deque()  
  
**from** tabulate **import** tabulate  
eps = **"Є"**LL1 = **True**firstDict = {}  
followDict = {}  
**def** CaluclateFirst(start):  
 first = set()  
  
 **for** v **in** rules[start]:  
 **for** each **in** v.split(**" "**):  
  
 **if** eps **in** first:  
 first.remove(eps)  
  
 **if** each **in** terminals:  
 first.add(each)  
  
 **elif** each == eps:  
 first.add(eps)  
  
 **elif** each **in** non\_terminals:  
 first = first.union(CaluclateFirst(each))  
  
 **if** eps **not in** first:  
 **break** firstDict[start] = first  
 **return** first  
  
  
**def** CalculateFollow(start):  
 follow = set()  
 **if** start == start\_symbol:  
 follow.add(**"$"**)  
 **for** k, r **in** rules.items():  
  
 **for** each **in** r:  
 each = each.split(**" "**)  
 **if** start **in** each:  
 index = each.index(start)

**if** index != len(each)-1:  
 **for** i **in** range(index+1, len(each)):  
 **if** eps **in** follow:  
 follow.remove(eps)  
 followDict[start] = follow  
 **if** each[i] **in** terminals:  
 follow = follow.union(each[i])  
 followDict[start] = follow  
 **else**:  
 follow = follow.union(firstDict[each[i]])  
 followDict[start] = follow  
 **if** eps **not in** follow:  
 **break  
 else**:  
 **if** k **not in** followDict:  
 follow = follow.union(CalculateFollow(k))  
 followDict[start] = follow  
 **else**:  
 follow = follow.union(followDict[k])  
 followDict[start] = follow  
 **if** eps **in** follow **or** len(follow) == 0:  
 **if** eps **in** follow:  
 follow.remove(eps)  
 follow = follow.union(CalculateFollow(k))  
 followDict[start] = follow  
 followDict[start] = follow  
 **return** follow  
  
  
*#input terminal symbols*print(**"=============================================================================================="**)  
print(**"Enter Terminals:"**)  
terminals = list(map(str, input().split()))  
  
*#input non terminal symbols*print(**"=============================================================================================="**)  
print(**"Enter Non- Terminals:"**)  
non\_terminals = list(map(str, input().split()))

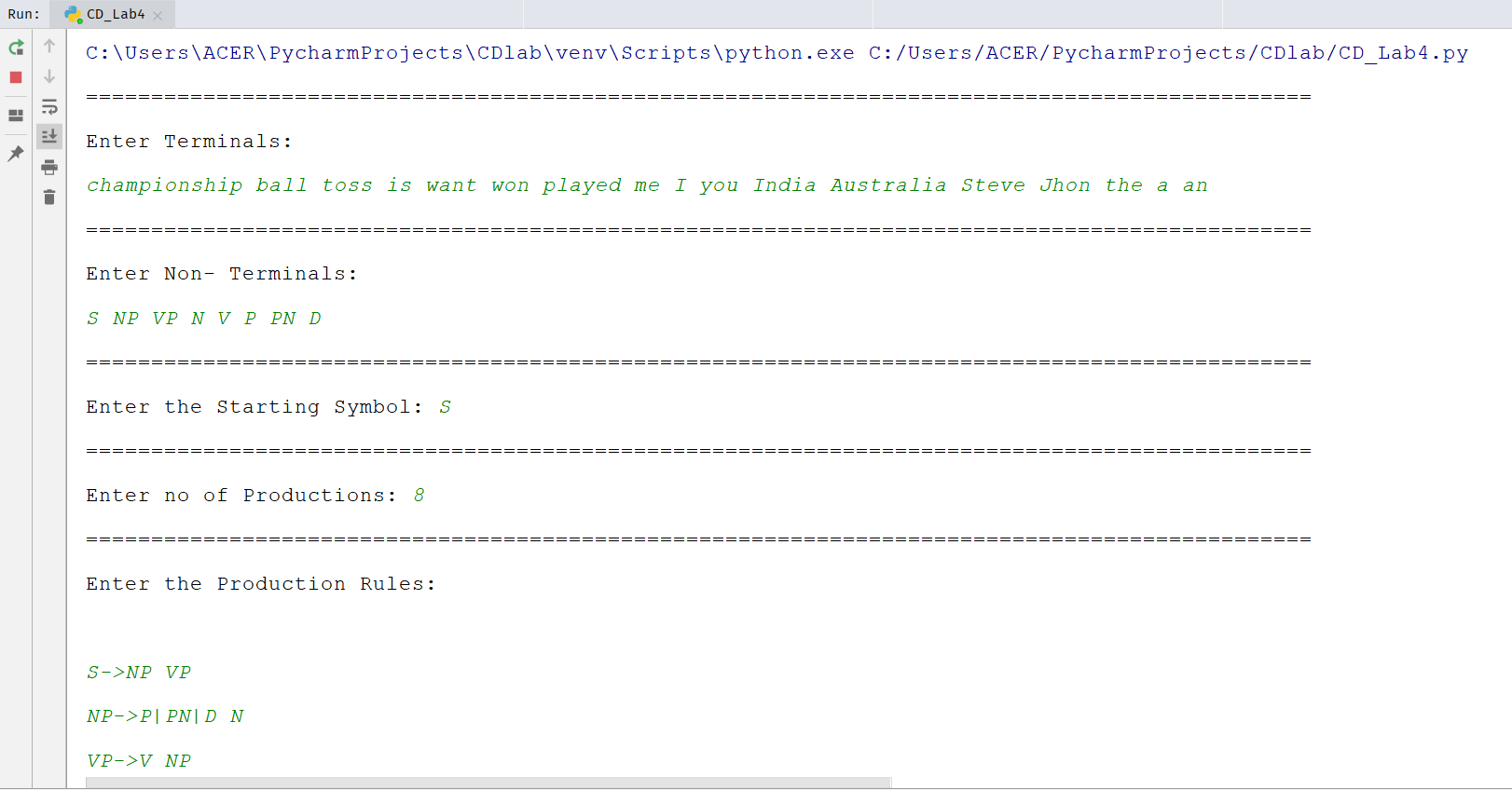
*#input start symbol*print(**"=============================================================================================="**)  
start\_symbol = input(**"Enter the Starting Symbol: "**)  
  
*#input all production rules*print(**"=============================================================================================="**)  
no\_of\_productions = int(input(**"Enter no of Productions: "**))  
productions = []  
print(**"=============================================================================================="**)  
print(**"Enter the Production Rules:\n"**)  
**for** \_ **in** range(no\_of\_productions):  
 productions.append(input().replace(**"#"**, eps))  
  
rules = {}  
**for** p **in** productions:  
 r = p.split(**"->"**)  
 rules[r[0]] = r[1].split(**'|'**)  
  
*#print(rules)*print(**"=============================================================================================="**)  
print(**"\t\t\t\tLL(1) PARSER STRING VALIDATION"**)  
*#print("GRAMMER :",rules)***for** start **in** non\_terminals:  
 CaluclateFirst(start)  
*#print(firstDict)***for** start **in** non\_terminals:  
 CalculateFollow(start)  
*#print(followDict)*tab = []  
print(**"=============================================================================================="**)  
print(**"\nFIRST & FOLLOW SET COMPUTATION TABLE"**)  
**for** FR, FL **in** zip(firstDict, followDict):  
 tab.append([FR, firstDict[FR], followDict[FR]])  
tab = tabulate(tab, headers=[**"SYMBOL"**, **"FIRST SET"**, **"FOLLOW SET"**])  
print(tab)

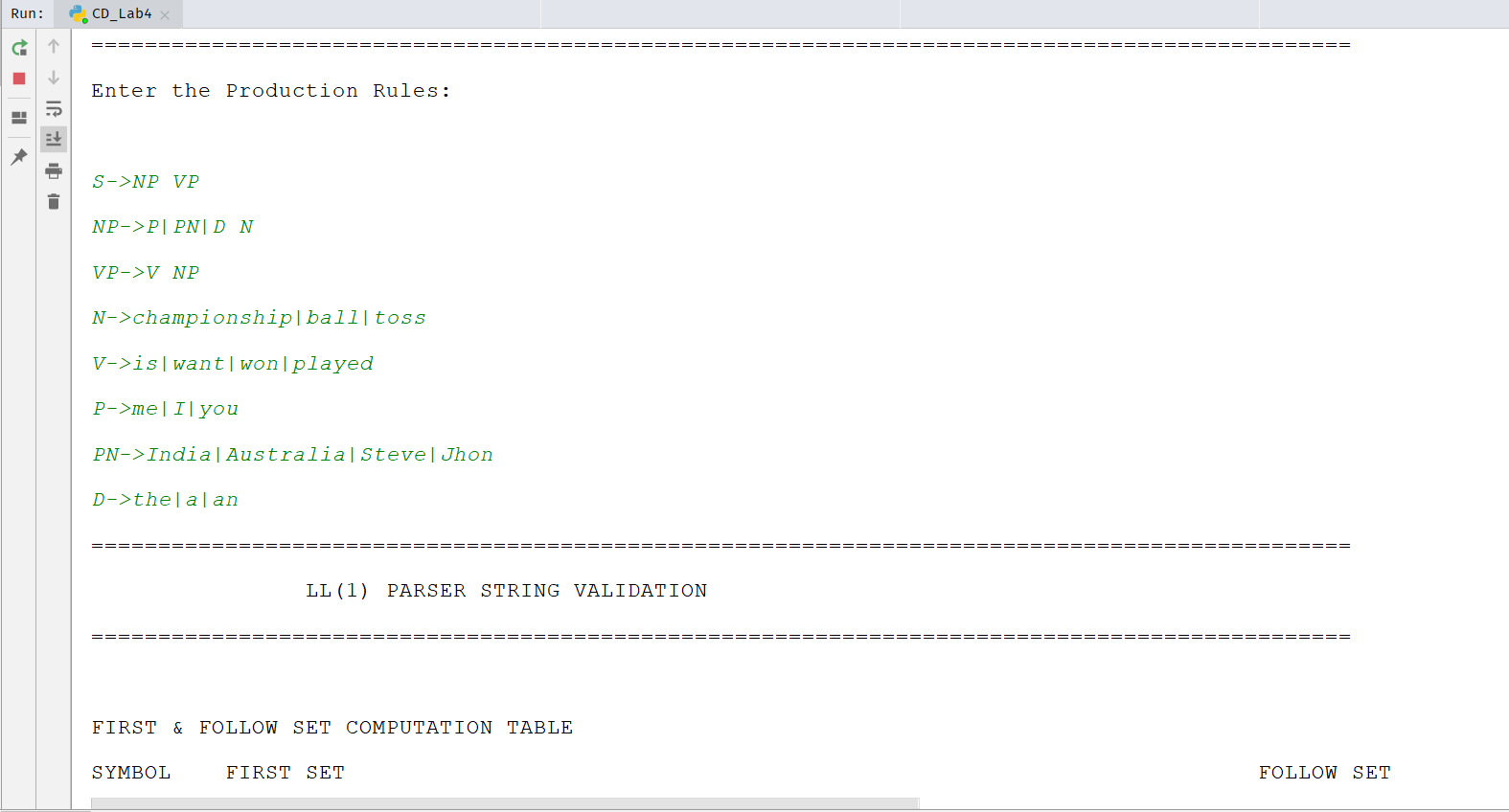
**def** parsingTable(rules):  
 **for** symbol, prod **in** rules.items():  
 **for** each **in** prod:  
 each = each.split(**" "**)  
 *#print(each)* t = set()  
 **for** e **in** each:  
 **if** e **in** non\_terminals:  
 **if** eps **in** t:  
 t.remove(eps)  
 t = t.union(firstDict[e])  
 **if** eps **not in** t:  
 **break  
 else**:  
 **if** eps **in** t:  
 t.remove(eps)  
  
 t = t.union([e])  
 **break  
  
 if** eps **in** t:  
 t.remove(eps)  
 t = t.union(followDict[symbol])  
 table[symbol].append([{symbol+**'->'**+**" "**.join(each): t}])  
  
  
  
  
table = dict()  
**for** each **in** non\_terminals:  
 table[each] = []  
  
parsingTable(rules)  
  
print(**"=============================================================================================="**)  
print(**"\nPARSING TABLE"**)  
tab = dict()  
  
d = dict()  
**for** t **in** terminals:  
 d[t] = []  
d[**"$"**] = []  
l = []  
  
  
**for** row **in** table:  
 NT, value = row, table[row]  
 **for** entry **in** value:  
 **for** cell **in** entry:  
 **for** item **in** cell:  
 **for** v **in** cell[item]:  
 d[v].append(item)  
 **if** len(d[v]) > 1:  
 LL1 = **False** tab[NT] = d  
 *#print(tab)* lst = list(d.values())  
 lst.insert(0, NT)  
 l.append(lst)  
 **for** t **in** terminals:  
 d[t] = []  
 d[**"$"**] = []  
 tab.clear()  
  
terminals.insert(0, **"SYMBOL"**)  
terminals.append(**"$"**)  
  
PT = tabulate(l, headers = terminals)  
print(PT)  
  
  
**def** invalid():  
 **global** stack  
 **global** buffer  
 stack = deque()  
 buffer = deque()  
 print(**"\n\t\t"**,string,**": Invalid String!"**)  
  
print(**"=============================================================================================="**)  
**if** LL1:  
 **while True**:  
 string = input(**"\n\nEnter String to Validate:"**)  
 print()  
 stack = deque()  
 stack.append(**"$"**)  
 stack.append(start\_symbol)  
 **if** string == **'0'**:  
 exit()

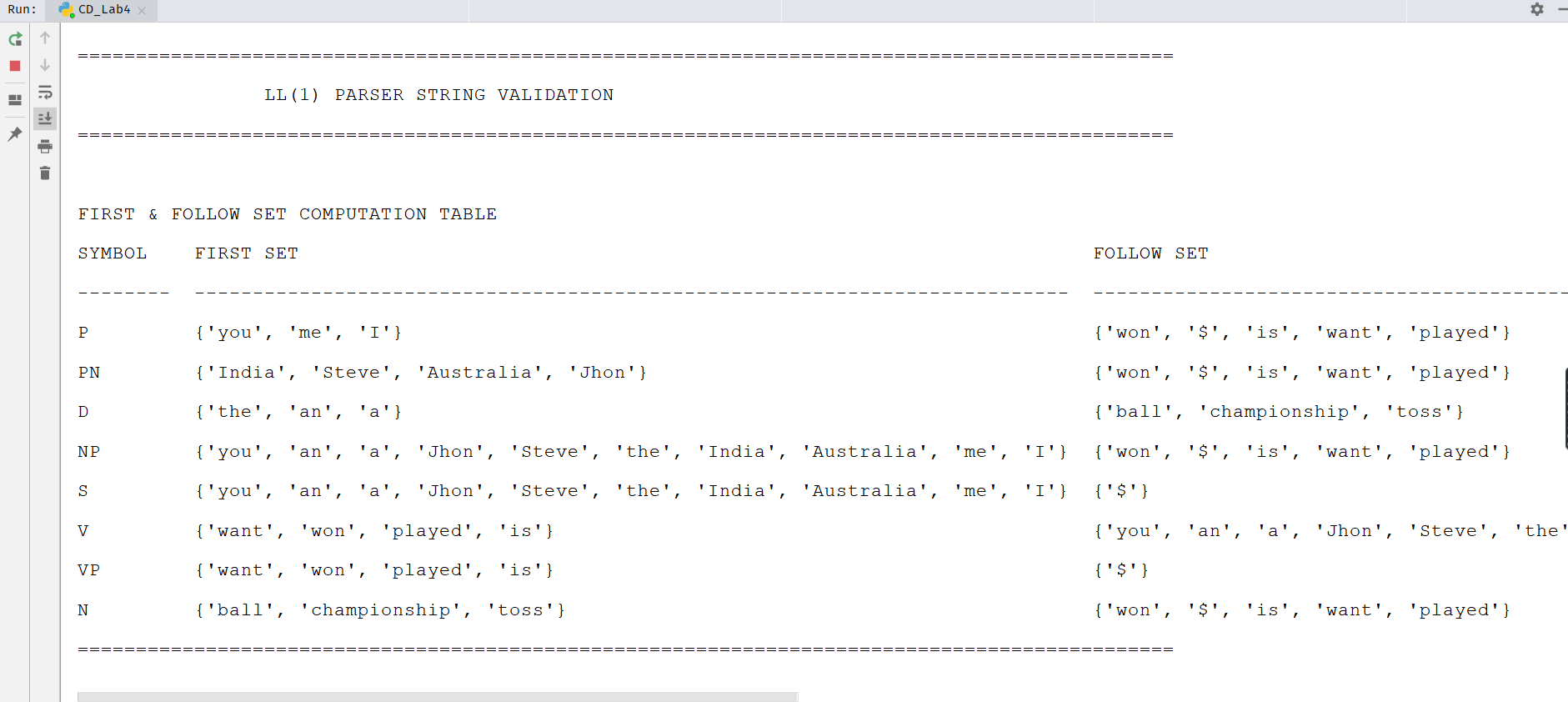
buffer = string.split()  
 buffer = buffer[::-1]  
 buffer.insert(0, **"$"**)  
  
 **while**(len(stack)>0 **and** len(buffer)>0):  
 print(buffer, stack)  
 Stop = stack.pop()  
 Btop = buffer[len(buffer)-1]  
 **if** Btop **in** terminals:  
 ind = terminals.index(Btop)  
 **else**:  
 *#print("3")* invalid()  
 **break** print(**'\t'**,Stop, Btop, end=**'\t'**)  
 **for** row **in** l:  
 **if** row[0] == Stop:  
 rule = row[ind]  
 print(**"Rule:"**,rule)  
 **if** rule == []:  
 *#print("2")* invalid()  
 **break** rule = **" "**.join(rule)  
 rule = rule.split(**'->'**)  
 rule = rule[1].split(**" "**)  
 **for** NT **in** rule[::-1]:  
 **if** NT == eps:  
 **continue** stack.append(NT)  
 Stop = stack[len(stack)-1]  
 *#print(buffer, stack)* **while**(Stop == Btop):  
  
 **if** Stop == **'$'**:  
 print(Stop, Btop)  
 print(**"\n\t\t"**, string, **": String is valid and accepted!"**)  
 stack = deque()  
 buffer = deque()  
 **break** print(**'\t'**,Stop, Btop, **'->Match'**)  
 stack.pop()  
 buffer.pop()  
 Stop = stack[len(stack)-1]  
 Btop = buffer[len(buffer)-1]

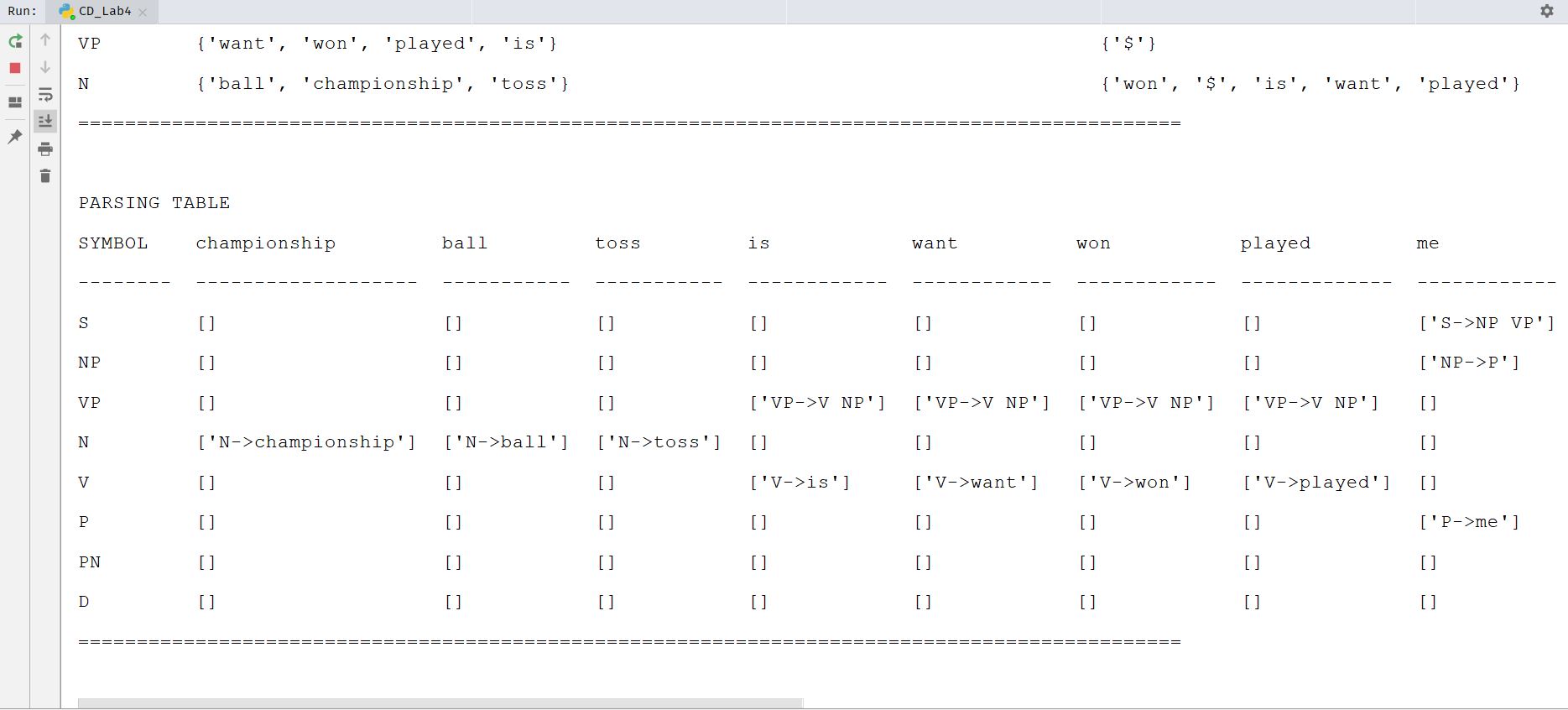
**if** Stop **in** terminals **and** Btop **in** terminals **and** Stop != Btop:  
 print(Stop, Btop)  
 *#print("1")* invalid()  
 **break  
 break**

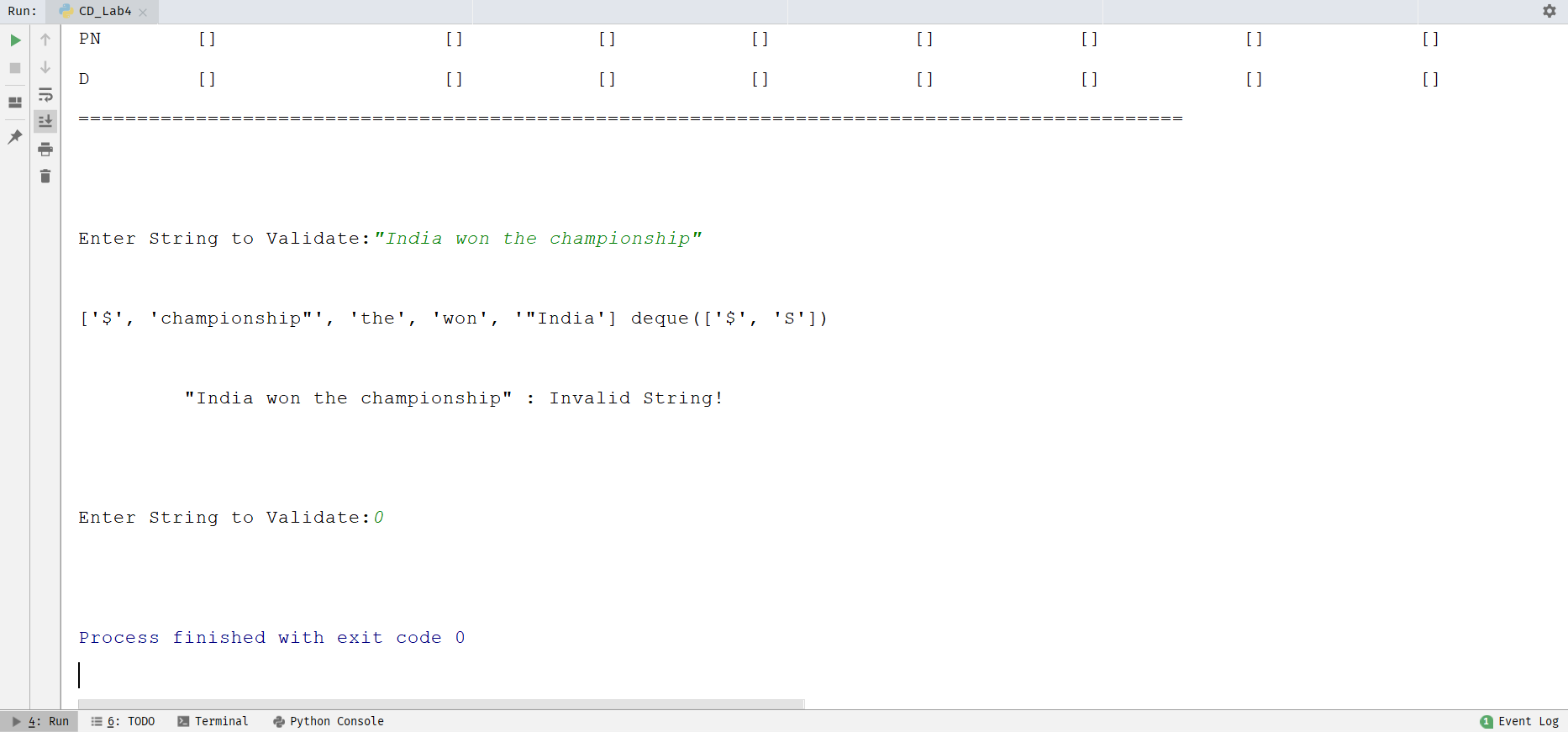
**OUTPUT :**



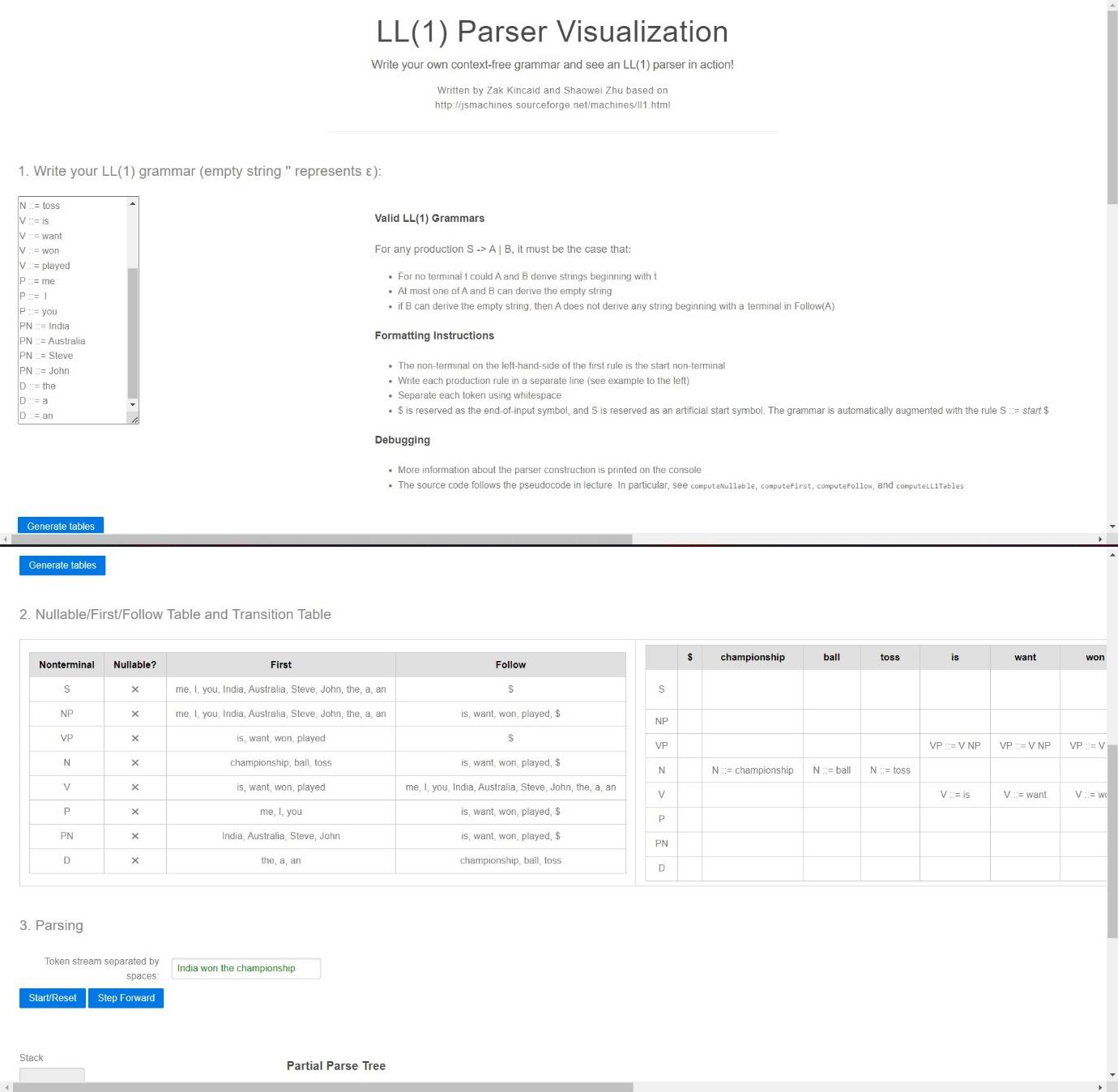








1. **Use Virtual Lab on LL1 parser to validate the string and verify your string validation using simulation.**







Link for Virtual Lab: <http://vlabs.iitb.ac.in/vlabs-> dev/vlab\_bootcamp/bootcamp/system\_deligators/labs/exp2/index.php

Use the below link for Pract 4 part (B) https://[www.cs.princeton.edu/courses/archive/spring20/cos320/LL1](http://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1)