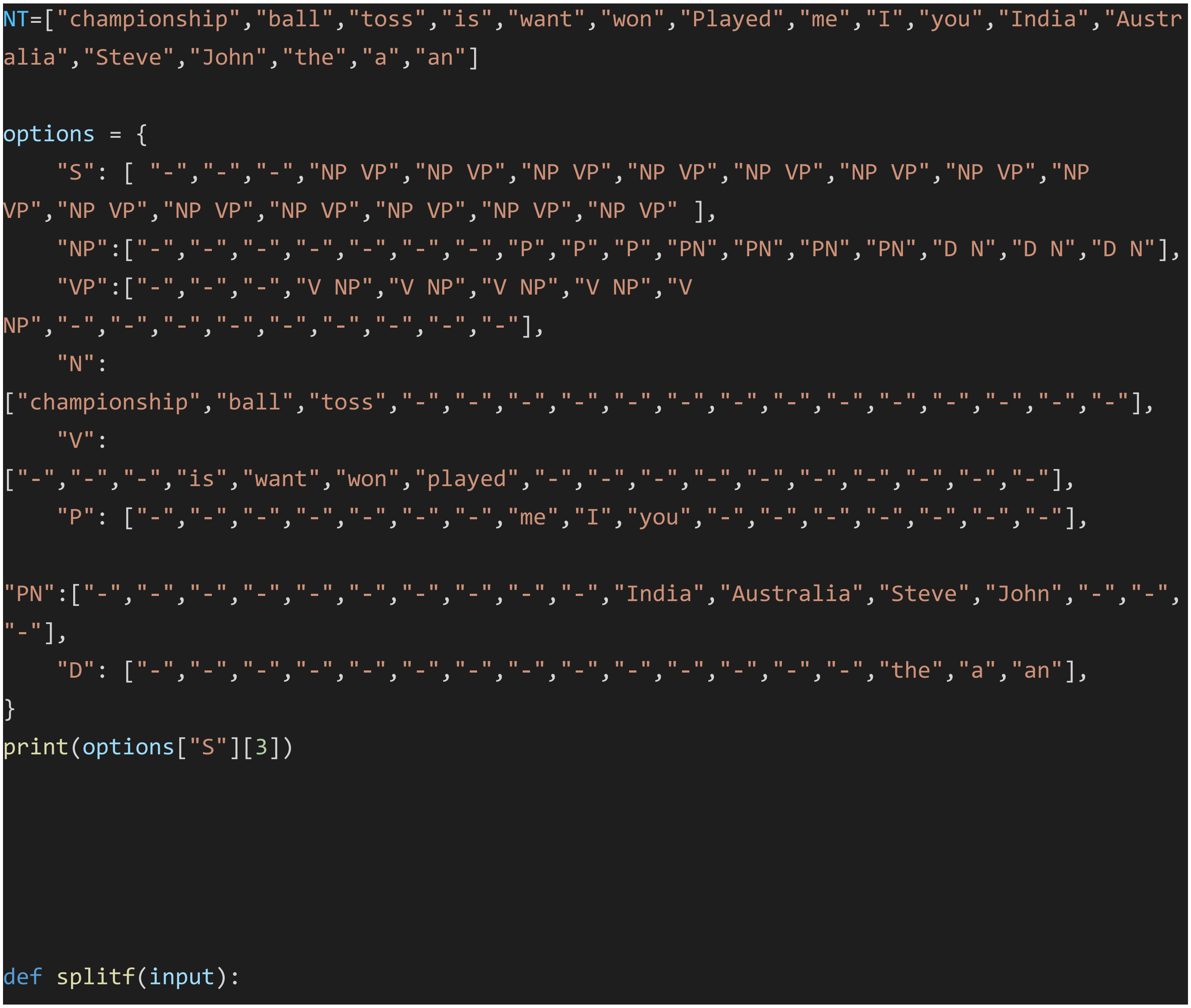
Practical - 04

Aim:

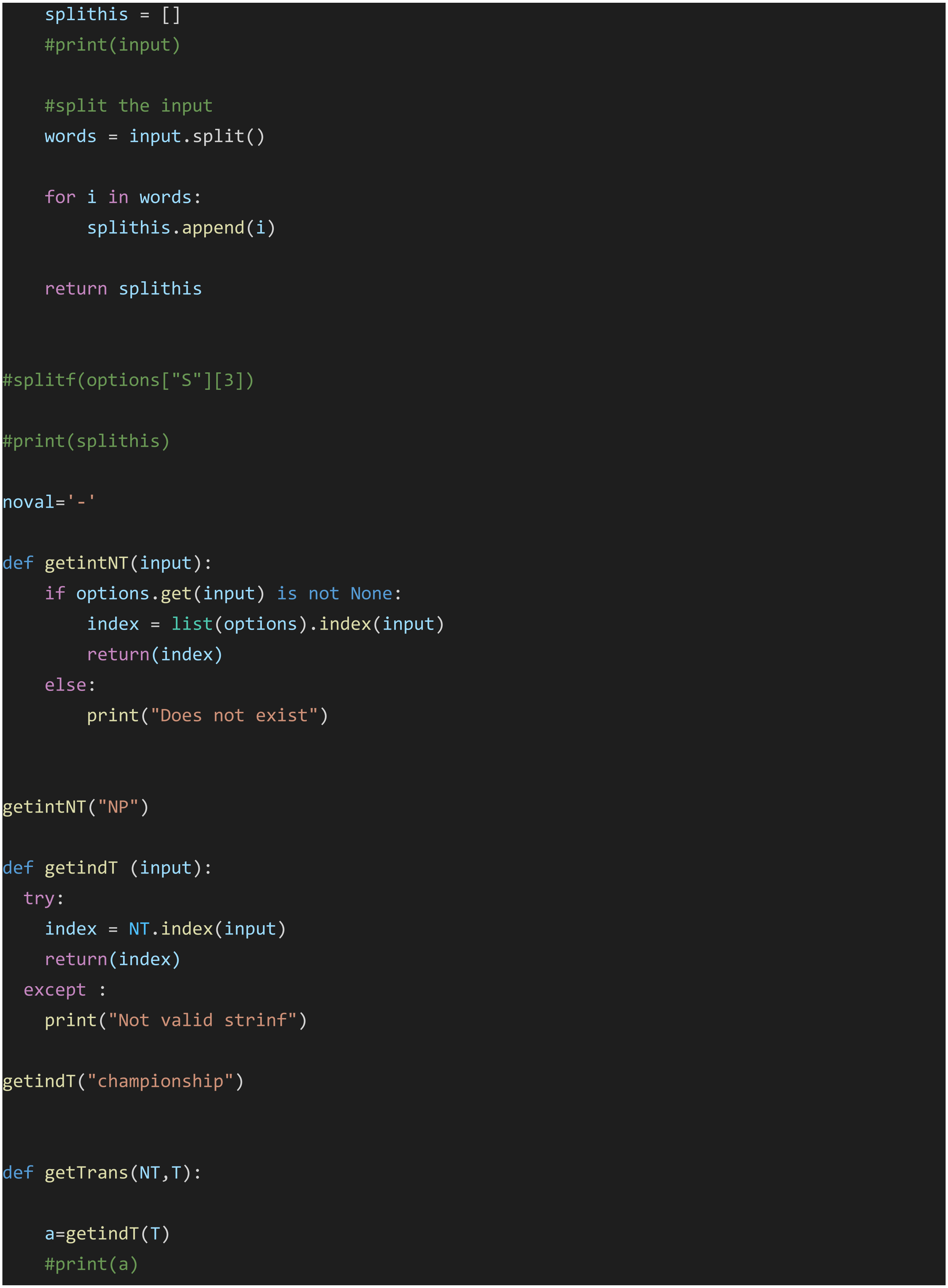
(A) Write a program to validate a natural language sentence. Design a natural language grammar, compute and input the LL(1) table. Validate if the given sentence is valid or not based on the grammar.

(B) Use Virtual Lab on LL1 parser to validate the string and verify your string validation using simulation.

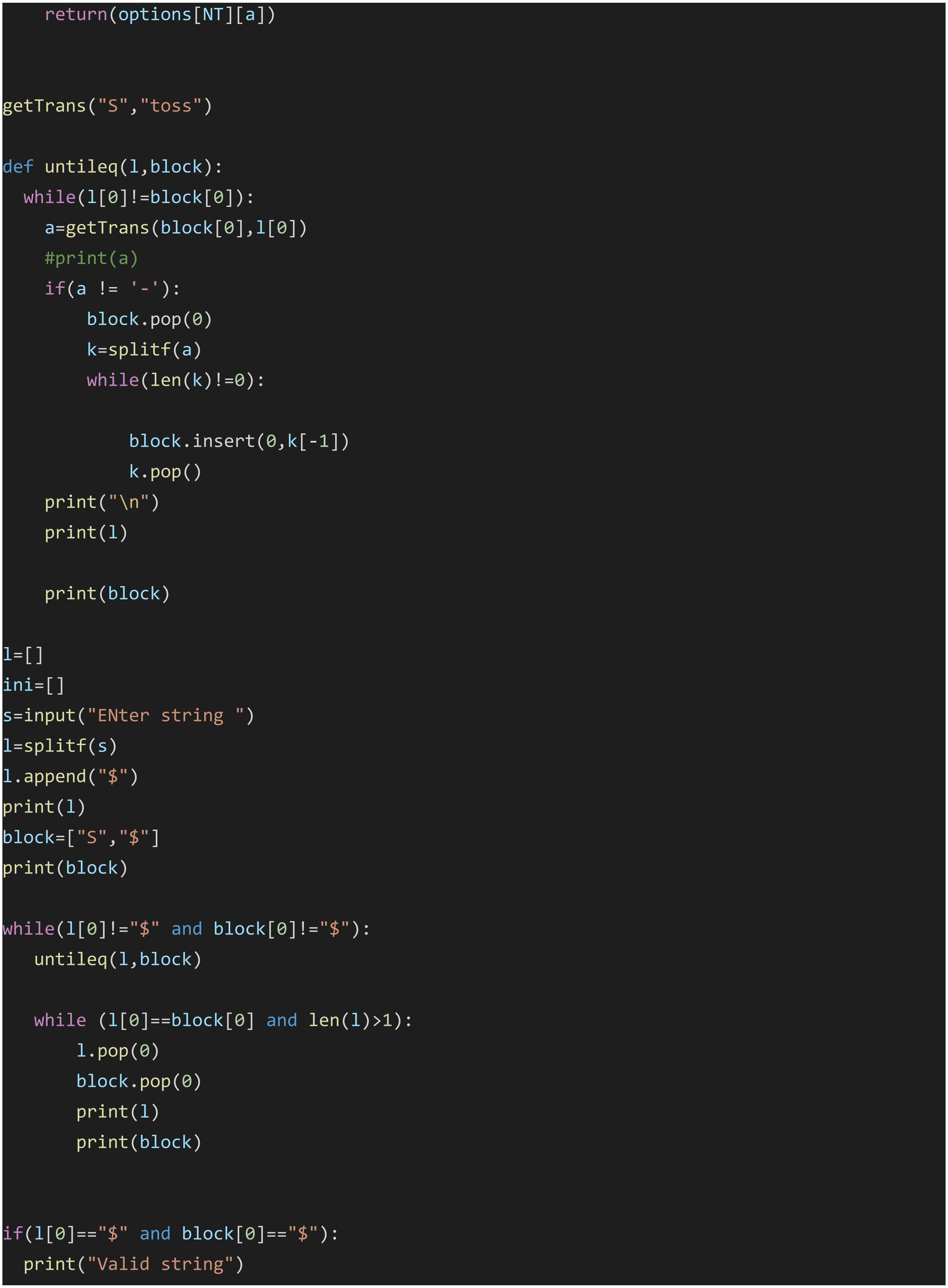
Code:-



```
NT=["championship","ball","toss","is","want","won","Played","me","I","you","India","Austr
alia","Steve","John","the","a","an"]

options = {
    "S": [ "-","-","-","NP VP","NP VP","NP VP","NP VP","NP VP","NP VP","NP VP","NP
VP","NP VP","NP VP","NP VP","NP VP","NP VP","NP VP" ],
    "NP":["-","-","-","-","-","-","-","P","P","P","PN","PN","PN","PN","D N","D N","D N"],
    "VP":["-","-","-","V NP","V NP","V NP","V NP","V
NP","-","-","-","-","-","-","-","-","-"],
    "N":
["championship","ball","toss","-","-","-","-","-","-","-","-","-","-","-","-","-","-"],
    "V":
["-","-","-","is","want","won","played","-","-","-","-","-","-","-","-","-","-"],
    "P": ["-","-","-","-","-","-","-","me","I","you","-","-","-","-","-","-","-"],

"PN":["-","-","-","-","-","-","-","-","-","-","India","Australia","Steve","John","-","-",
"-"],
    "D": ["-","-","-","-","-","-","-","-","-","-","-","-","-","-","the","a","an"],
}
print(options["S"][3])




def splitf(input):
```



```
    splithis = []
    #print(input)

    #split the input
    words = input.split()

    for i in words:
        splithis.append(i)

    return splithis


#splitf(options["S"][3])

#print(splithis)

noval='-'

def getintNT(input):
    if options.get(input) is not None:
        index = list(options).index(input)
        return(index)
    else:
        print("Does not exist")


getintNT("NP")

def getindT (input):
  try:
    index = NT.index(input)
    return(index)
  except :
    print("Not valid strinf")

getindT("championship")


def getTrans(NT,T):

    a=getindT(T)
    #print(a)
```



```
    return(options[NT][a])


getTrans("S","toss")

def untileq(l,block):
  while(l[0]!=block[0]):
    a=getTrans(block[0],l[0])
    #print(a)
    if(a != '-'):
        block.pop(0)
        k=splitf(a)
        while(len(k)!=0):

            block.insert(0,k[-1])
            k.pop()
    print("\n")
    print(l)

    print(block)

l=[]
ini=[]
s=input("ENter string ")
l=splitf(s)
l.append("$")
print(l)
block=["S","$"]
print(block)

while(l[0]!="$" and block[0]!="$"):
   untileq(l,block)

   while (l[0]==block[0] and len(l)>1):
       l.pop(0)
       block.pop(0)
       print(l)
       print(block)


if(l[0]=="$" and block[0]=="$"):
  print("Valid string")
```

```
else:
  print("Not valid string”)
```

Output:-

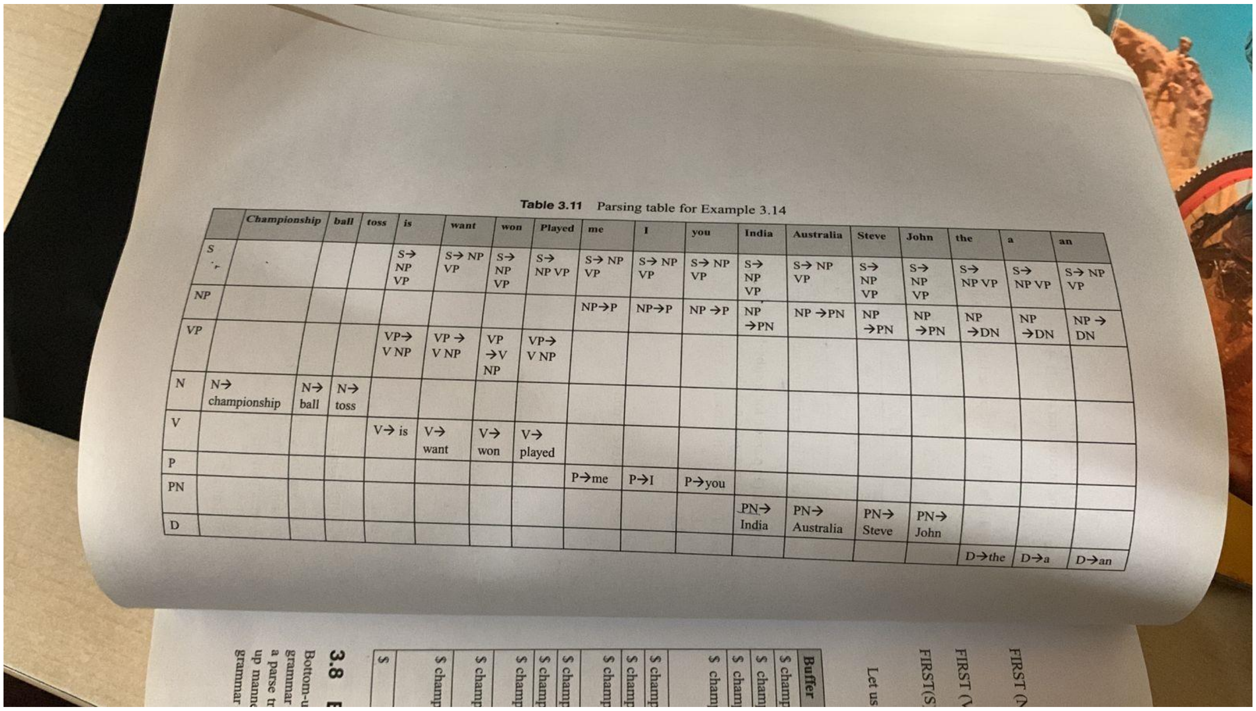
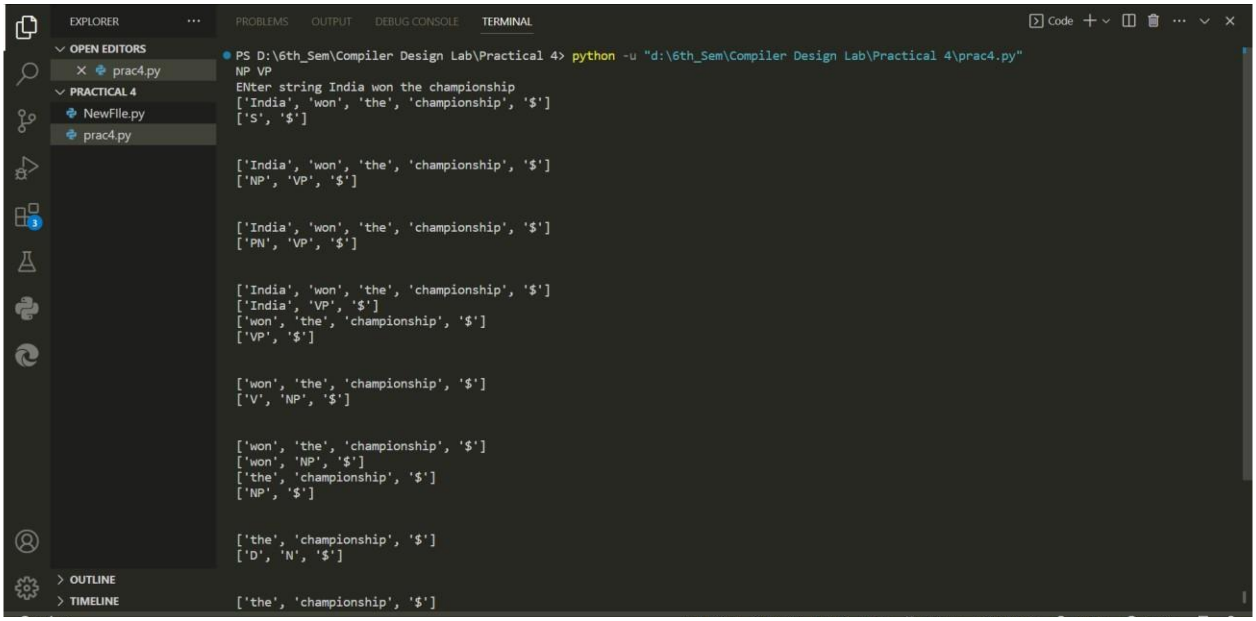
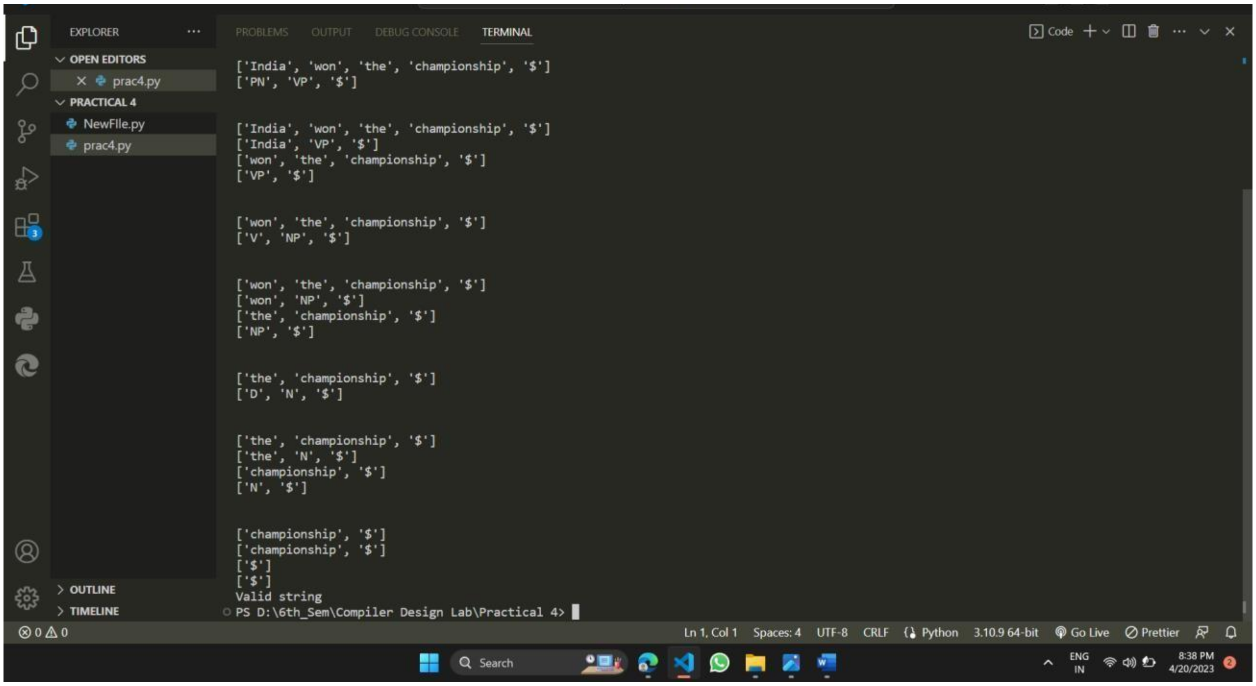


Table 3.11 Parsing table for Example 3.14

	Championship	ball	toss	is	want	won	Played	me	I	you	India	Australia	Steve	John	the	a	an
S				S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP	S→ NP VP
NP								NP→P	NP→P	NP →P	NP →PN	NP →PN	NP →PN	NP →PN	NP →DN	NP →DN	NP → DN
VP				VP→ V NP	VP → V NP	VP →V NP	VP→ V NP										
N	N→ championship	N→ ball	N→ toss														
V				V→ is	V→ want	V→ won	V→ played										
P								P→me	P→I	P→you							
PN											PN→ India	PN→ Australia	PN→ Steve	PN→ John			
D															D→the	D→a	D→an



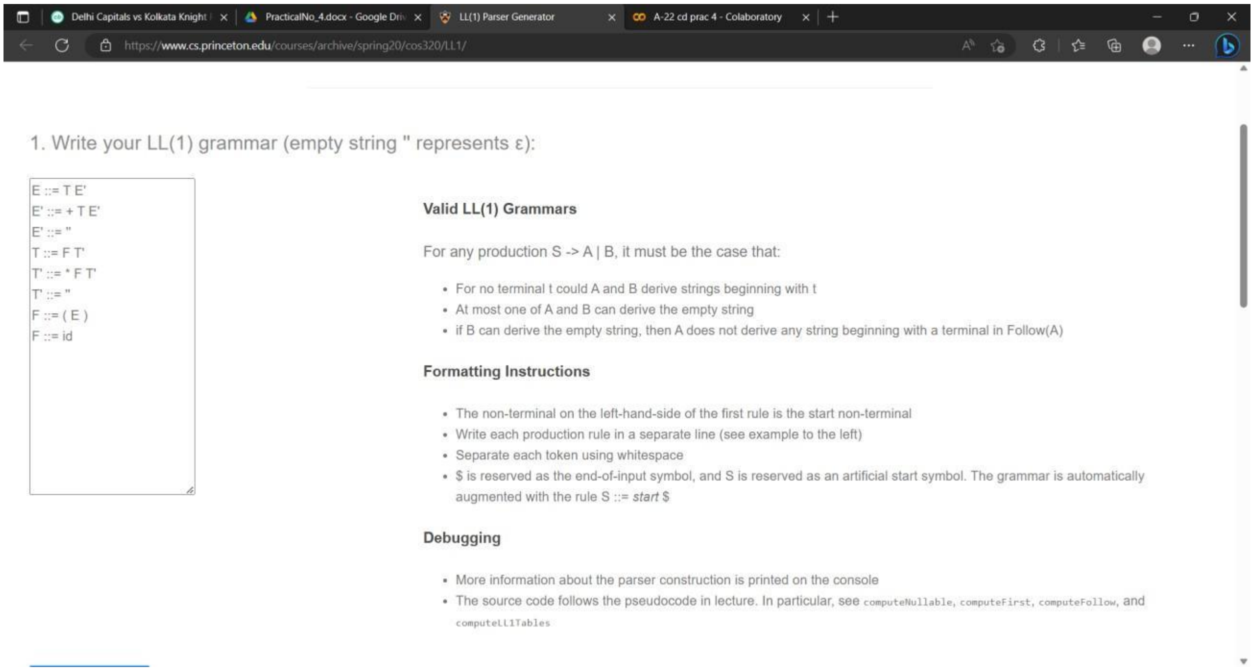
```
PS D:\6th_Sem\Compiler Design Lab\Practical 4> python -u "d:\6th_Sem\Compiler Design Lab\Practical 4\prac4.py"
NP VP
ENter string India won the championship
['India', 'won', 'the', 'championship', '$']
['S', '$']


['India', 'won', 'the', 'championship', '$']
['NP', 'VP', '$']


['India', 'won', 'the', 'championship', '$']
['PN', 'VP', '$']


['India', 'won', 'the', 'championship', '$']
['India', 'VP', '$']
['won', 'the', 'championship', '$']
['VP', '$']


['won', 'the', 'championship', '$']
['V', 'NP', '$']


['won', 'the', 'championship', '$']
['won', 'NP', '$']
['the', 'championship', '$']
['NP', '$']


['the', 'championship', '$']
['D', 'N', '$']


['the', 'championship', '$']
```



```
['India', 'won', 'the', 'championship', '$']
['PN', 'VP', '$']


['India', 'won', 'the', 'championship', '$']
['India', 'VP', '$']
['won', 'the', 'championship', '$']
['VP', '$']


['won', 'the', 'championship', '$']
['V', 'NP', '$']


['won', 'the', 'championship', '$']
['won', 'NP', '$']
['the', 'championship', '$']
['NP', '$']


['the', 'championship', '$']
['D', 'N', '$']


['the', 'championship', '$']
['the', 'N', '$']
['championship', '$']
['N', '$']


['championship', '$']
['championship', '$']
['$']
['$']
Valid string
PS D:\6th_Sem\Compiler Design Lab\Practical 4>
```

(A) Part

Screenshots:


Delhi Capitals vs Kolkata Knight
PracticalNo_4.docx - Google Driv
LL(1) Parser Generator
A-22 cd prac 4 - Colaboratory
https://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1/

1. Write your LL(1) grammar (empty string '' represents ε):

```
E ::= T E'
E' ::= + T E'
E' ::= ''
T ::= F T'
T' ::= * F T'
T' ::= ''
F ::= ( E )
F ::= id
```

Valid LL(1) Grammars

For any production S -> A | B, it must be the case that:

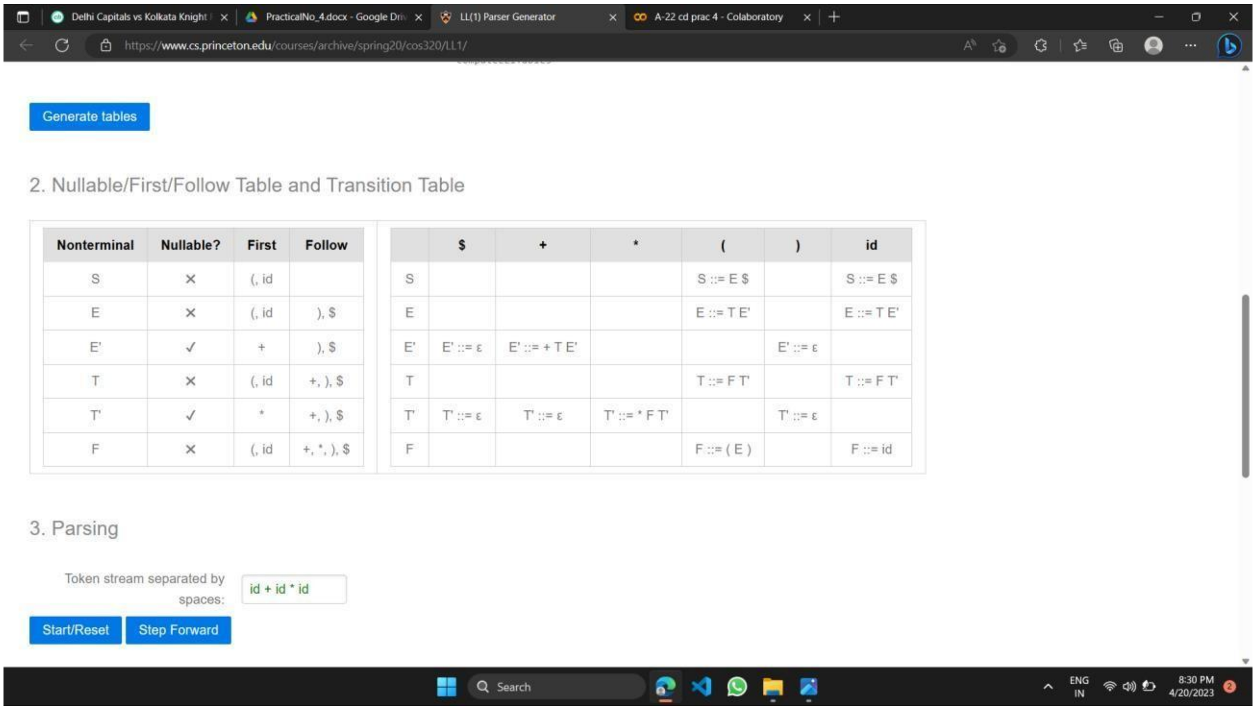
- For no terminal t could A and B derive strings beginning with t
- At most one of A and B can derive the empty string
- if B can derive the empty string, then A does not derive any string beginning with a terminal in Follow(A)

Formatting Instructions

- The non-terminal on the left-hand-side of the first rule is the start non-terminal
- Write each production rule in a separate line (see example to the left)
- Separate each token using whitespace
- $ is reserved as the end-of-input symbol, and S is reserved as an artificial start symbol. The grammar is automatically augmented with the rule S ::= *start* $

Debugging

- More information about the parser construction is printed on the console
- The source code follows the pseudocode in lecture. In particular, see `computeNullable`, `computeFirst`, `computeFollow`, and `computeLL1Tables`


Delhi Capitals vs Kolkata Knight
PracticalNo_4.docx - Google Driv
LL(1) Parser Generator
A-22 cd prac 4 - Colaboratory
https://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1/
Generate tables
Search
ENG IN
8:30 PM 4/20/2023

2. Nullable/First/Follow Table and Transition Table

Nonterminal	Nullable?	First	Follow
S	×	(, id	
E	×	(, id	), $
E'	✓	+	), $
T	×	(, id	+,), $
T'	✓	*	+,), $
F	×	(, id	+, *,), $

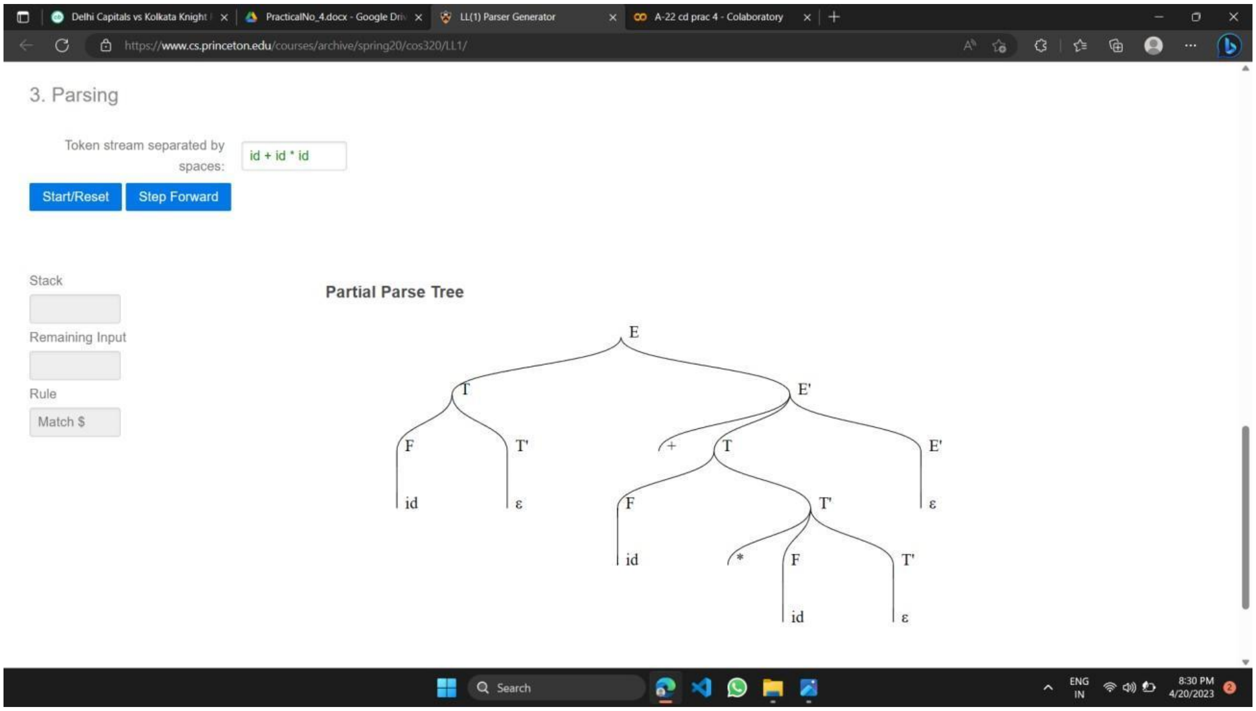
	$	+	*	(	)	id
S				S ::= E $		S ::= E $
E				E ::= T E'		E ::= T E'
E'	E' ::= ε	E' ::= + T E'			E' ::= ε	
T				T ::= F T'		T ::= F T'
T'	T' ::= ε	T' ::= ε	T' ::= * F T'		T' ::= ε	
F				F ::= (E)		F ::= id

3. Parsing

Token stream separated by spaces: id + id * id

Start/Reset Step Forward

Parsing Tree


Delhi Capitals vs Kolkata Knight
PracticalNo_4.docx - Google Driv
LL(1) Parser Generator
A-22 cd prac 4 - Colaboratory
https://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1/
E
T
E'
F
T'
+
T
E'
id
ε
F
T'
ε
id
*
F
T'
id
ε
Search
ENG IN
8:30 PM 4/20/2023

3. Parsing

Token stream separated by spaces: id + id * id

Start/Reset Step Forward

Stack

Remaining Input

Rule

Match $

Partial Parse Tree