Name: Harsh Darji
Assignment 2

In this problem, you need to develop the context-free grammar that can parse all the following sentences. You will use Python environment to test your grammar to make sure that it does the job and save your Python screenshot. (45%) [Hint: the lab instructions on POS tagging (lab 5) and Context-Free Grammar (lab 6) may be useful]

(a) "Today is a nice day"

(b) "Bob and Mary went to France last month again"

(c) "You can say that again"

(d) "Birds are not necessarily able to fly"

Solution:

Start Symbol: S

Non-terminal symbol: VP, NP, R, JJ

Terminal Symbol: NN, NNP, NNS, CC, JJ, DT, TO, VBZ, VBD, VB, RB, PRP, MD

Grammar:

```
my_grammar = nltk.PCFG.fromstring("""
  S -> NP VP
  VP -> VBD TO NP | VB DT R | VB | VBZ DT J | VBZ R
  R -> RB R J | RB
  J -> JJ NP | JJ TO VP
  NP -> NNP | NN R | NNP JJ NP | NNP CC NP | NN | NNS | PRP MD
  NN -> "Today" |"day" | "month"
  NNP -> "Mary" | "France" | "Bob"
  NNS -> "Birds"
  CC -> "and"
  JJ -> "last" | "able" |"nice"
  DT -> "a" | "that"
  TO -> "to"
  VBZ -> "is" | "are"
```

Name: Harsh Darji
Assignment 2

VBD -> "went"

VB -> "fly" | "say"

RB -> "not" | "necessarily" | "again"

PRP -> "You"

MD -> "can"

""")

a. 'Today is a nice day'

Output:

```
(S (NP (NN Today)) (VP (VBZ is) (DT a) (J (JJ nice) (NP (NN day)))))
```

Screenshot:

```
rd_parser = nltk.RecursiveDescentParser(my_grammar)
```

```
sent5list = 'Today is a nice day'.split()
for tree in rd_parser.parse(sent5list):
   print (tree)
```

```
(S (NP (NN Today)) (VP (VBZ is) (DT a) (J (JJ nice) (NP (NN day)))))
```

b. 'Bob and Mary went to France last month again'

Output:

```
(S
  (NP (NNP Bob) (CC and) (NP (NNP Mary)))
  (VP
    (VBD went)
    (TO to)
    (NP (NNP France) (JJ last) (NP (NN month) (R (RB again))))))
```

Screenshot:

Name: Harsh Darji
Assignment 2

```
sent4list= 'Bob and Mary went to France last month again'.split()
for tree in rd_parser.parse(sent4list):
   print (tree)
```

```
(S
  (NP (NNP Bob) (CC and) (NP (NNP Mary)))
  (VP
    (VBD went)
    (TO to)
    (NP (NNP France) (JJ last) (NP (NN month) (R (RB again))))))
```

c. 'You can say that again'

Output:

```
(S (NP (PRP You) (MD can)) (VP (VB say) (DT that) (R (RB again))))
```

Screenshot:

```
]: sent3list= 'You can say that again'.split()
   for tree in rd_parser.parse(sent3list):
      print (tree)

      (S (NP (PRP You) (MD can)) (VP (VB say) (DT that) (R (RB again))))
```

d. 'Birds are not necessarily able to fly'

Output:

```
(S
  (NP (NNS Birds))
  (VP
    (VBZ are)
    (R
      (RB not)
      (R (RB necessarily))
      (J (JJ able) (TO to) (VP (VB fly))))))
```

Screenshot:

Name: Harsh Darji
Assignment 2

```
sent2list= 'Birds are not necessarily able to fly'.split()
for tree in rd_parser.parse(sent2list):
   print (tree)
```

```
(S
  (NP (NNS Birds))
  (VP
    (VBZ are)
    (R
      (RB not)
      (R (RB necessarily))
      (J (JJ able) (TO to) (VP (VB fly))))))
```

Besides these four sentences, list up to three different sentences that can be parsed by this grammar as well as the corresponding Python screenshot. Of the three sentences, can you generate one that does not make sense? Why can't you have such a sentence based on your grammar? (15%)

a. **'Bob and Mary say that again'**

Output:

```
(S
  (NP (NNP Bob) (CC and) (NP (NNP Mary)))
  (VP (VB say) (DT that) (R (RB again))))
```

Screenshot:

```
sent2list= 'Bob and Mary say that again'.split()
for tree in rd_parser.parse(sent2list):
   print (tree)
```

```
(S
  (NP (NNP Bob) (CC and) (NP (NNP Mary)))
  (VP (VB say) (DT that) (R (RB again))))
```

b. 'Birds went to France last month again'

Output:

```
(S
  (NP (NNS Birds))
  (VP
    (VBD went)
    (TO to)
```

Name: Harsh Darji
Assignment 2

```
    (NP (NNP France) (JJ last) (NP (NN month) (R (RB again))))))
```

Screenshot:

```
      (VP (VB say) (DT that) (R (RB again))))
```

```
: sent2list= 'Birds went to France last month again'.split()
for tree in rd_parser.parse(sent2list):
   print (tree)
```

```
(S
  (NP (NNS Birds))
  (VP
    (VBD went)
    (TO to)
    (NP (NNP France) (JJ last) (NP (NN month) (R (RB again))))))
```

c. 'went Today to France'

Output:

Could not parse this sentence. This sentence does not follow the rules give by the grammar. Sentence is incomplete. Also, we don't have rules defined by our grammar to parse this sentence. The Grammar defined is strict and it precisely follow rules.

Screenshot:

```
sent2list= 'went Today to France'.split()
for tree in rd_parser.parse(sent2list):
   print (tree)
```

Now, assuming that the above four sentences are your mini-mini training corpus, you will write a probabilistic context-free grammar. You will use Python environments to test your grammar and save the Python screenshot. (40%)

prob_grammar = nltk.PCFG.fromstring("""

S -> NP VP [1.0]

VP -> | VBD TO NP [0.2] | VB DT R [0.2] | VB [0.2] | VBZ DT J [0.2] | VBZ R [0.2]

R -> RB R J [0.5] | RB [0.5]

J -> JJ NP [0.5] | JJ TO VP [0.5]

Name: Harsh Darji
Assignment 2

NP -> NNP [0.15] | NN R [0.15] | NNP JJ NP [0.1] | NNP CC NP [0.15] | NN [0.15] | NNS [0.15] | PRP MD [0.15]

NN -> "Today" [0.2] |"day" [0.4] | "month" [0.4]

NNP -> "Mary" [0.2] | "France" [0.4] | "Bob" [0.4]

NNS -> "Birds" [1.0]

CC -> "and" [1.0]

JJ -> "last" [0.4] | "able" [0.2] |"nice" [0.4]

DT -> "a" [0.5] | "that" [0.5]

TO -> "to" [1.0]

VBZ -> "is" [0.5] | "are" [0.5]

VBD -> "went" [1.0]

VB -> "fly" [0.5] | "say" [0.5]

RB -> "not" [0.25] | "necessarily" [0.25] | "again" [0.5]

PRP -> "You" [1.0]

MD -> "can" [1.0]

""")

Screenshot:

```
prob_grammar = nltk.PCFG.fromstring("""
    S -> NP VP [1.0]
    VP -> | VBD TO NP [0.2] | VB DT R [0.2] | VB [0.2] | VBZ DT J [0.2] | VBZ R [0.2]
    R ->  RB R J [0.5] | RB [0.5]
    J -> JJ NP [0.5] | JJ TO VP [0.5]
    NP -> NNP [0.15] | NN R [0.15] | NNP JJ NP [0.1] | NNP CC NP [0.15] | NN [0.15] | NNS [0.15] | PRP MD [0.15]
    NN -> "Today" [0.2] |"day" [0.4] | "month" [0.4]
    NNP -> "Mary" [0.2] | "France" [0.4] | "Bob" [0.4]
    NNS -> "Birds" [1.0]
    CC -> "and" [1.0]
    JJ -> "last" [0.4] | "able" [0.2] |"nice" [0.4]
    DT -> "a" [0.5] | "that" [0.5]
    TO -> "to" [1.0]
    VBZ -> "is" [0.5] | "are" [0.5]
    VBD -> "went" [1.0]
    VB -> "fly" [0.5] | "say" [0.5]
    RB -> "not" [0.25] | "necessarily" [0.25] | "again" [0.5]
    PRP -> "You" [1.0]
    MD -> "can" [1.0]

    """)
```

Sentences:

Name: Harsh Darji
Assignment 2

a. **'Today is a nice day'**

Output:

```
(S (NP (NN Today)) (VP (VBZ is) (DT a) (J (JJ nice) (NP (NN day)))))
(p=1.8e-05)
```

Screenshot:

```
v_parser = nltk.ViterbiParser(prob_grammar)
```

```
sent5list = 'Today is a nice day'.split()
for tree in v_parser.parse(sent5list):
   print (tree)
```

```
(S
  (NP (NN Today))
  (VP (VBZ is) (DT a) (J (JJ nice) (NP (NN day))))) (p=1.8e-05)
```

b. **"Bob and Mary went to France last month again"**

Output:

```
(S
  (NP (NNP Bob) (CC and) (NP (NNP Mary)))
  (VP
    (VBD went)
    (TO to)
    (NP
      (NNP France)
      (JJ last)
      (NP (NN month) (R (RB again)))))) (p=8.64e-08)
```

Screenshot:

```
sent4list= 'Bob and Mary went to France last month again'.split()
for tree in v_parser.parse(sent4list):
   print (tree)
```

```
(S
  (NP (NNP Bob) (CC and) (NP (NNP Mary)))
  (VP
    (VBD went)
    (TO to)
    (NP
      (NNP France)
      (JJ last)
      (NP (NN month) (R (RB again)))))) (p=8.64e-08)
```

Name: Harsh Darji
Assignment 2

c. "You can say that again"

Output:

```
(S
  (NP (PRP You) (MD can))
  (VP (VB say) (DT that) (R (RB again)))) (p=0.001875)
```

Screenshot:

```
sent3list= 'You can say that again'.split()
for tree in v_parser.parse(sent3list):
    print (tree)
```

```
(S
  (NP (PRP You) (MD can))
  (VP (VB say) (DT that) (R (RB again)))) (p=0.001875)
```

d. "Birds are not necessarily able to fly"

Output:

```
(S
  (NP (NNS Birds))
  (VP
    (VBZ are)
    (R
      (RB not)
      (R (RB necessarily))
      (J (JJ able) (TO to) (VP (VB fly)))))) (p=2.34375e-06)
```

Screenshot:

Name: Harsh Darji
Assignment 2

```
sent2list= 'Birds are not necessarily able to fly'.split()
for tree in v_parser.parse(sent2list):
   print (tree)
```

```
(S
  (NP (NNS Birds))
  (VP
    (VBZ are)
    (R
      (RB not)
      (R (RB necessarily))
      (J (JJ able) (TO to) (VP (VB fly))))))) (p=2.34375e-06)
```