I. Context-free grammer that can parse all the following sentences:

Context-free grammer

```
my_grammar = nltk.CFG.fromstring("""
  S -> WRB SQ | NP VP | VP | NP ADVP VP
  WRB -> "Why"
  JJ -> "last"
  NN -> "month"
  RB -> "again" | "not" | "always"
  UH -> "Please"
  PRP -> "their"
  INTJ -> UH
  SQ -> V NP VP
  VP -> V NP | V PP NP ADVP | INTJ VP2 ADVP
  ADVP -> RB
  PP -> To NP
  V -> "do" | "have" | "went" | "chase"
  NP -> Det | Prop | NNP CC NNP | NNP | JJ NN | N | PRP N
  VP2 -> V RB VP3
  VP3 -> V NP
  Prop -> "you"
  NNP -> "Bob" | "Mary" | "France"
  Det -> "this" | "that"
  N -> "Dogs" | "tails"
  To -> "to"
  CC -> "and"
  """)
```

a). “Why do you have this?”

First, we define a recursive descent parser from above grammar.

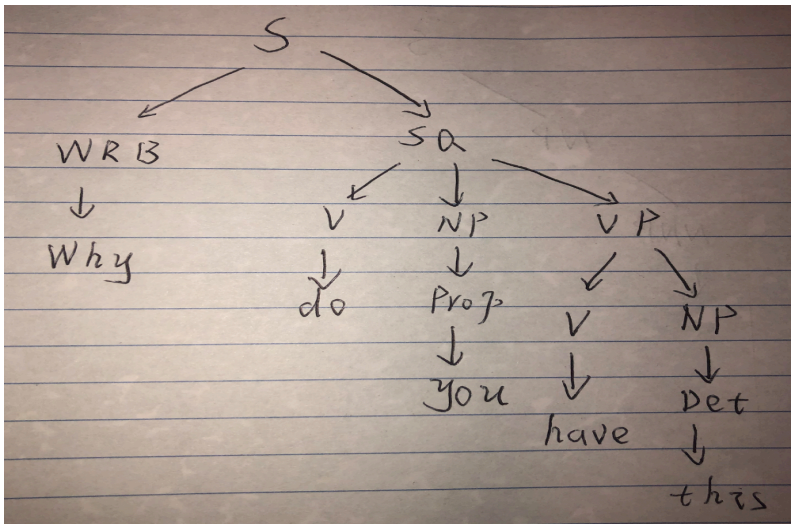
```
rd_parser = nltk.RecursiveDescentParser(my_grammar)
```

Then we produce a list of tokens that were separated by white space.

```
sentlist1 = 'Why do you have this'.split()
```

Then we use the parser to return a generator for the list of all trees that it found.

```
for tree in rd_parser.parse(sentlist1):
    print (tree)
```




```
(S
  (WRB Why)
  (SQ (V do) (NP (Prop you)) (VP (V have) (NP (Det this)))))
```

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

First, we define a recursive descent parser from above grammar.

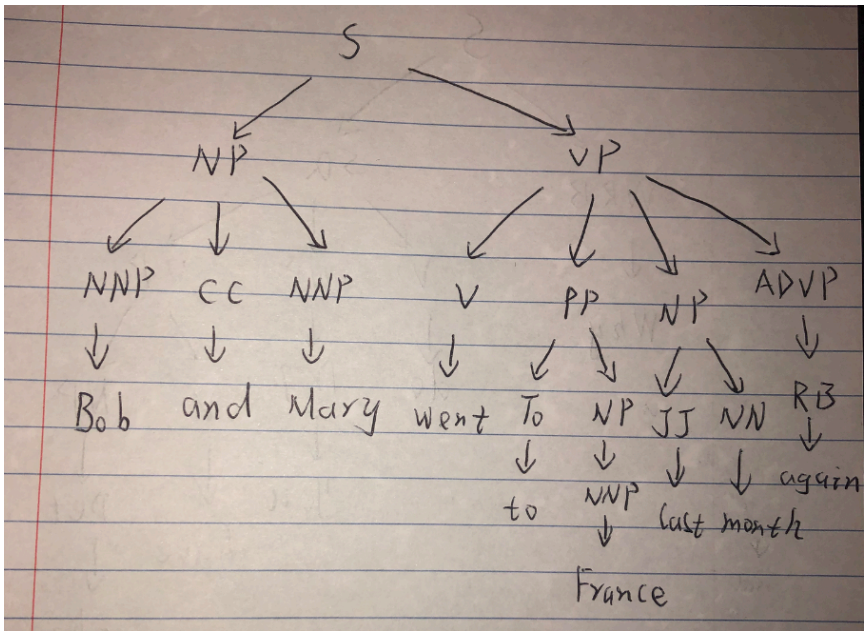
```
rd_parser = nltk.RecursiveDescentParser(my_grammar)
```

Then we produce a list of tokens that were separated by white space.

```
sentlist2 = 'Bob and Mary went to France last month again'.split()
```

Then we use the parser to return a generator for the list of all trees that it found.

```
for tree in rd_parser.parse(sentlist2):
    print (tree)
```




```
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again))))
```

c). "Please do not do that again!"

First, we define a recursive descent parser from above grammar.

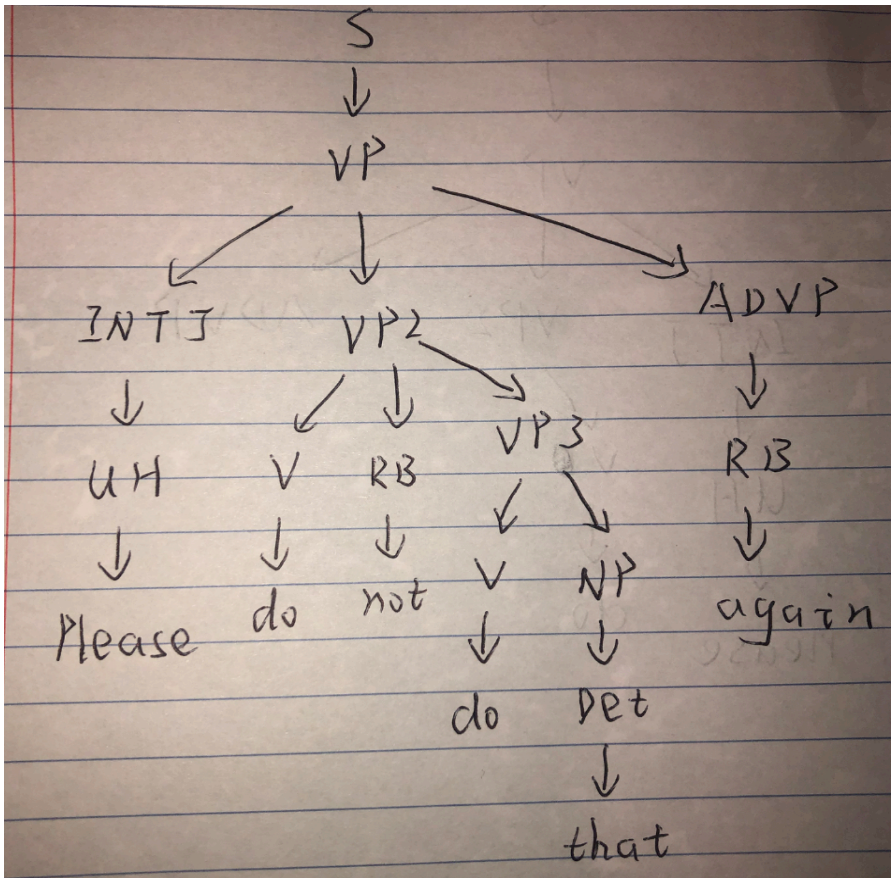
```
rd_parser = nltk.RecursiveDescentParser(my_grammar)
```

Then we produce a list of tokens that were separated by white space.

```
sentlist3 = 'Please do not do that again'.split()
```

Then we use the parser to return a generator for the list of all trees that it found.

```
for tree in rd_parser.parse(sentlist3):
    print (tree)
```




```
(S
  (VP
    (INTJ (UH Please))
    (VP2 (V do) (RB not) (VP3 (V do) (NP (Det that))))
    (ADVP (RB again))))
```

d). "Dogs always chase their tails."

First, we define a recursive descent parser from above grammar.

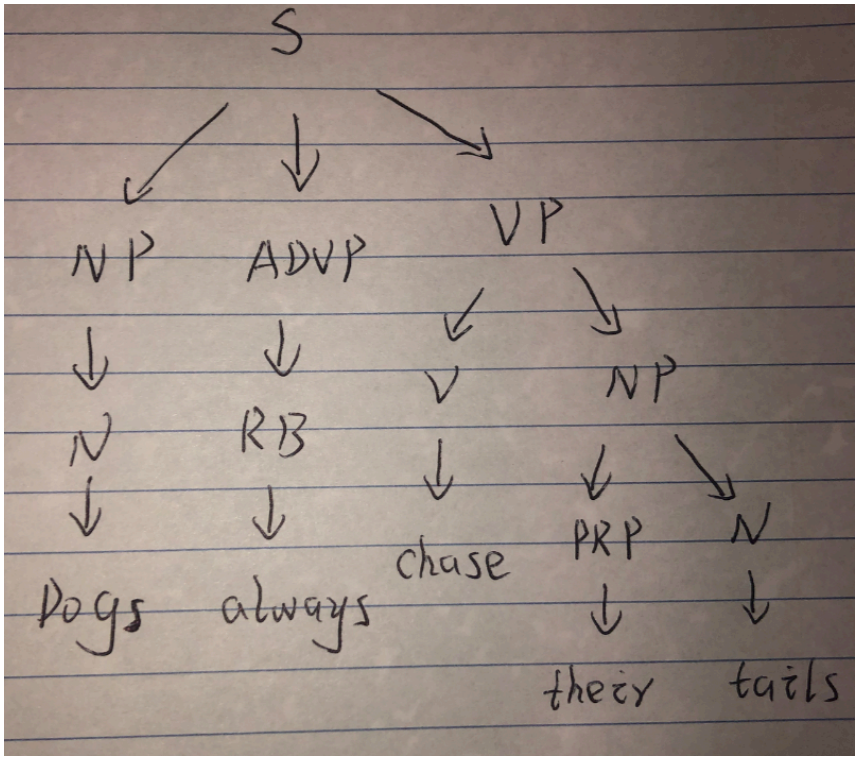
```
rd_parser = nltk.RecursiveDescentParser(my_grammar)
```

Then we produce a list of tokens that were separated by white space.

```
sentlist4 = 'Dogs always chase their tails'.split()
```

Then we use the parser to return a generator for the list of all trees that it found.

```
for tree in rd_parser.parse(sentlist4):
    print (tree)
```




```
(S
  (NP (N Dogs))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

II. Three different sentences which can be parsed by this grammer:

a). Dogs went to France last month again.

```
(S
  (NP (N Dogs))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again))))
```

b). Bob and Mary always chase their tails.

```
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

c). France always chase their tails.

```
(S
  (NP (NNP France))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

III. Probabilistic context-free grammer

I get the probabilistic by following code:

```
from nltk import induce_pcfg
from nltk import Nonterminal
S = Nonterminal('root')
productions = []
for tree in rd_parser.parse(sentlist1):
    productions += tree.productions()
for tree in rd_parser.parse(sentlist2):
    productions += tree.productions()
for tree in rd_parser.parse(sentlist3):
    productions += tree.productions()
for tree in rd_parser.parse(sentlist4):
    productions += tree.productions()
grammar = induce_pcfg(S, productions)
print(grammar)
```

The probabilistic is like following:

```
Grammar with 35 productions (start state = root)
    S -> NP ADVP VP [0.5]
    NP -> NNP [0.25]
    NNP -> 'France' [0.5]
    ADVP -> RB [1.0]
    RB -> 'always' [0.4]
    VP -> V NP [0.5]
    V -> 'chase' [0.4]
    NP -> PRP N [0.25]
    PRP -> 'their' [1.0]
    N -> 'tails' [0.666667]
    S -> NP VP [0.25]
    NP -> NNP CC NNP [0.125]
    NNP -> 'Bob' [0.25]
    CC -> 'and' [1.0]
```

```
    NNP -> 'Mary' [0.25]
    VP -> V PP NP ADVP [0.25]
    V -> 'went' [0.2]
    PP -> To NP [1.0]
    To -> 'to' [1.0]
    NP -> JJ NN [0.125]
    JJ -> 'last' [1.0]
    NN -> 'month' [1.0]
    RB -> 'again' [0.4]
    S -> VP [0.25]
    VP -> INTJ VP2 ADVP [0.25]
    INTJ -> UH [1.0]
    UH -> 'Please' [1.0]
    VP2 -> V RB VP3 [1.0]
```

```
  V -> 'do' [0.4]
  RB -> 'not' [0.2]
  VP3 -> V NP [1.0]
  NP -> Det [0.125]
  Det -> 'that' [1.0]
  NP -> N [0.125]
  N -> 'Dogs' [0.333333]
```

Following is the probabilistic grammer:

```
prob_grammar = nltk.PCFG.fromstring("""
  S -> WRB SQ[0.25] | NP VP[0.25]| VP[0.25] | NP ADVP VP[0.25]
  WRB -> "Why"[1.0]
  JJ -> "last"[1.0]
  NN -> "month"[1.0]
  RB -> "again"[0.5] | "not"[0.25] | "always"[0.25]
  UH -> "Please"[1.0]
  PRP -> "their"[1.0]
  INTJ -> UH[1.0]
  SQ -> V NP VP[1.0]
  VP -> V NP[0.5] | V PP NP ADVP[0.25] | INTJ VP2 ADVP[0.25]
  ADVP -> RB[1.0]
  PP -> To NP[1.0]
  V -> "do"[0.5] | "have"[0.166667] | "went"[0.166667] | "chase"[0.166666]
  NP -> Det[0.25] | Prop[0.125] | NNP CC NNP[0.125] | NNP[0.125] | JJ NN[0.125] | N[0.125] | PRP N[0.125]
  VP2 -> V RB VP3[1.0]
  VP3 -> V NP[1.0]
  Prop -> "you"[1.0]
  NNP -> "Bob"[0.333333] | "Mary"[0.333333] | "France"[0.333334]
  Det -> "this"[0.5] | "that"[0.5]
  N -> "Dogs"[0.5] | "tails"[0.5]
  To -> "to"[1.0]
  CC -> "and"[1.0]
  """)
```

a). “Why do you have this?” (result is as following)

```
(S
  (WRB Why)
  (SQ (V do) (NP (Prop you)) (VP (V have) (NP (Det this)))))  (p=0.000162761)
```

b). “Bob and Mary went to France last month again.” (result is as following)

```
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again))))  (p=3.76761e-07)
```

c). “Please do not do that again!” (result is as following)

```
(S
  (VP
    (INTJ (UH Please))
    (VP2 (V do) (RB not) (VP3 (V do) (NP (Det that))))
    (ADVP (RB again))))  (p=0.000244141)
```

d). “Dogs always chase their tails.” (result is as following)

```
(S
  (NP (N Dogs))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails)))) (p=2.0345e-05)
```

IV: Appendix

Output:

Trees of the four sentences:

```
(S
  (WRB Why)
  (SQ (V do) (NP (Prop you)) (VP (V have) (NP (Det this)))))
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again))))
```

```
(S
  (VP
    (INTJ (UH Please))
    (VP2 (V do) (RB not) (VP3 (V do) (NP (Det that))))
    (ADVP (RB again))))
(S
  (NP (N Dogs))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

Three different sentences that can be parsed by this grammar

```
(S
  (NP (N Dogs))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again))))
```

```
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

```
(S
  (NP (NNP France))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

probabilistic context-free grammar

```
(S
  (WRB Why)
  (SQ (V do) (NP (Prop you)) (VP (V have) (NP (Det this)))))) (p=0.000162761)
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again)))) (p=3.76761e-07)
```

```
(S
  (VP
    (INTJ (UH Please))
    (VP2 (V do) (RB not) (VP3 (V do) (NP (Det that))))
    (ADVP (RB again)))) (p=0.000244141)
(S
  (NP (N Dogs))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails)))) (p=2.0345e-05)
```

V. Python process screenshot

```
>>> import nltk
>>> my_grammar = nltk.CFG.fromstring("""
...   S -> WRB SQ | NP VP | VP | NP ADVP VP
...   WRB -> "Why"
...   JJ -> "last"
...   NN -> "month"
...   RB -> "again" | "not" | "always"
...   UH -> "Please"
...   PRP -> "their"
...   INTJ -> UH
...   SQ -> V NP VP
...   VP -> V NP | V PP NP ADVP | INTJ VP2 ADVP
...   ADVP -> RB
...   PP -> To NP
...   V -> "do" | "have" | "went" | "chase"
...   NP -> Det | Prop | NNP CC NNP | NNP | JJ NN | N | PRP N
...   VP2 -> V RB VP3
...   VP3 -> V NP
...   Prop -> "you"
...   NNP -> "Bob" | "Mary" | "France"
...   Det -> "this" | "that"
...   N -> "Dogs" | "tails"
...   To -> "to"
...   CC -> "and"
...   """)
>>> rd_parser = nltk.RecursiveDescentParser(my_grammar)
>>> sentlist1 = 'Why do you have this'.split()
>>> for tree in rd_parser.parse(sentlist1):
...     print (tree)
...
(S
  (WRB Why)
  (SQ (V do) (NP (Prop you)) (VP (V have) (NP (Det this)))))


>>> sentlist2 = 'Bob and Mary went to France last month again'.split()
>>> for tree in rd_parser.parse(sentlist2):
...     print (tree)
...
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again))))
>>> sentlist3 = 'Please do not do that again'.split()
>>> for tree in rd_parser.parse(sentlist3):
...     print (tree)
...
(S
  (VP
    (INTJ (UH Please))
    (VP2 (V do) (RB not) (VP3 (V do) (NP (Det that))))
    (ADVP (RB again))))
```

```
>>> sentlist3 = 'Please do not do that again'.split()
>>> for tree in rd_parser.parse(sentlist3):
...     print (tree)
...
(S
  (VP
    (INTJ (UH Please))
    (VP2 (V do) (RB not) (VP3 (V do) (NP (Det that))))
    (ADVP (RB again))))
>>> sentlist4 = 'Dogs always chase their tails'.split()
>>> for tree in rd_parser.parse(sentlist4):
...     print (tree)
...
(S
  (NP (N Dogs))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

```
(S
  (NP (N Dogs))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

```
>>> sentlist5 = 'Dogs went to France last month again'.split()
>>> for tree in rd_parser.parse(sentlist5):
...     print (tree)
...
(S
  (NP (N Dogs))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again))))
>>> sentlist6 = 'Bob and Mary always chase their tails'.split()
>>> for tree in rd_parser.parse(sentlist6):
...     print (tree)
...
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
>>> sentlist7 = 'France always chase their tails'.split()
>>> for tree in rd_parser.parse(sentlist7):
...     print (tree)
...
(S
  (NP (NNP France))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails))))
```

```
>>> prob_grammar = nltk.PCFG.fromstring("""
...   S -> WRB SQ[0.25] | NP VP[0.25]| VP[0.25] | NP ADVP VP[0.25]
...   WRB -> "Why"[1.0]
...   JJ -> "last"[1.0]
...   NN -> "month"[1.0]
...   RB -> "again"[0.5] | "not"[0.25] | "always"[0.25]
...   UH -> "Please"[1.0]
...   PRP -> "their"[1.0]
...   INTJ -> UH[1.0]
...   SQ -> V NP VP[1.0]
...   VP -> V NP[0.5] | V PP NP ADVP[0.25] | INTJ VP2 ADVP[0.25]
...   ADVP -> RB[1.0]
...   PP -> To NP[1.0]
...   V -> "do"[0.5] | "have"[0.166667] | "went"[0.166667] | "chase"[0.166666]
...   NP -> Det[0.25] | Prop[0.125] | NNP CC NNP[0.125] | NNP[0.125] | JJ NN[0.1
25] | N[0.125] | PRP N[0.125]
...   VP2 -> V RB VP3[1.0]
...   VP3 -> V NP[1.0]
...   Prop -> "you"[1.0]
...   NNP -> "Bob"[0.333333] | "Mary"[0.333333] | "France"[0.333334]
...   Det -> "this"[0.5] | "that"[0.5]
...   N -> "Dogs"[0.5] | "tails"[0.5]
...   To -> "to"[1.0]
...   CC -> "and"[1.0]
...   """)
```

```
>>> def do_word_tokenize(content):
...     # do word tokenizing process
...     tokens = nltk.word_tokenize(content)
...     return tokens
...
>>> viterbi_parser = nltk.ViterbiParser(prob_grammar)
>>> content1 = "Why do you have this"
>>> tokens_content = do_word_tokenize(content1)
>>> for tree in viterbi_parser.parse(tokens_content):
...     print(tree)
...
(S
  (WRB Why)
  (SQ (V do) (NP (Prop you)) (VP (V have) (NP (Det this))))) (p=0.000162761)
>>> content2 = "Bob and Mary went to France last month again"
>>> tokens_content = do_word_tokenize(content2)
```

```
>>> tokens_content = do_word_tokenize(content2)
>>> for tree in viterbi_parser.parse(tokens_content):
...     print(tree)
...
(S
  (NP (NNP Bob) (CC and) (NNP Mary))
  (VP
    (V went)
    (PP (To to) (NP (NNP France)))
    (NP (JJ last) (NN month))
    (ADVP (RB again)))) (p=3.76761e-07)
>>> content3 = "Please do not do that again"
>>> tokens_content = do_word_tokenize(content3)
>>> for tree in viterbi_parser.parse(tokens_content):
...     print(tree)
...
(S
  (VP
    (INTJ (UH Please))
    (VP2 (V do) (RB not) (VP3 (V do) (NP (Det that))))
    (ADVP (RB again)))) (p=0.000244141)
>>> content4 = "Dogs always chase their tails"
>>> tokens_content = do_word_tokenize(content4)
>>> for tree in viterbi_parser.parse(tokens_content):
...     print(tree)
...
(S
  (NP (N Dogs))
  (ADVP (RB always))
  (VP (V chase) (NP (PRP their) (N tails)))) (p=2.0345e-05)
```