**CIS 668: Natural Language Processing**

**Assignment:3 Mahima Singh 454995544**

1. Developing context-free grammar that can parse all the following sentences.

(a) “Today is a nice day”

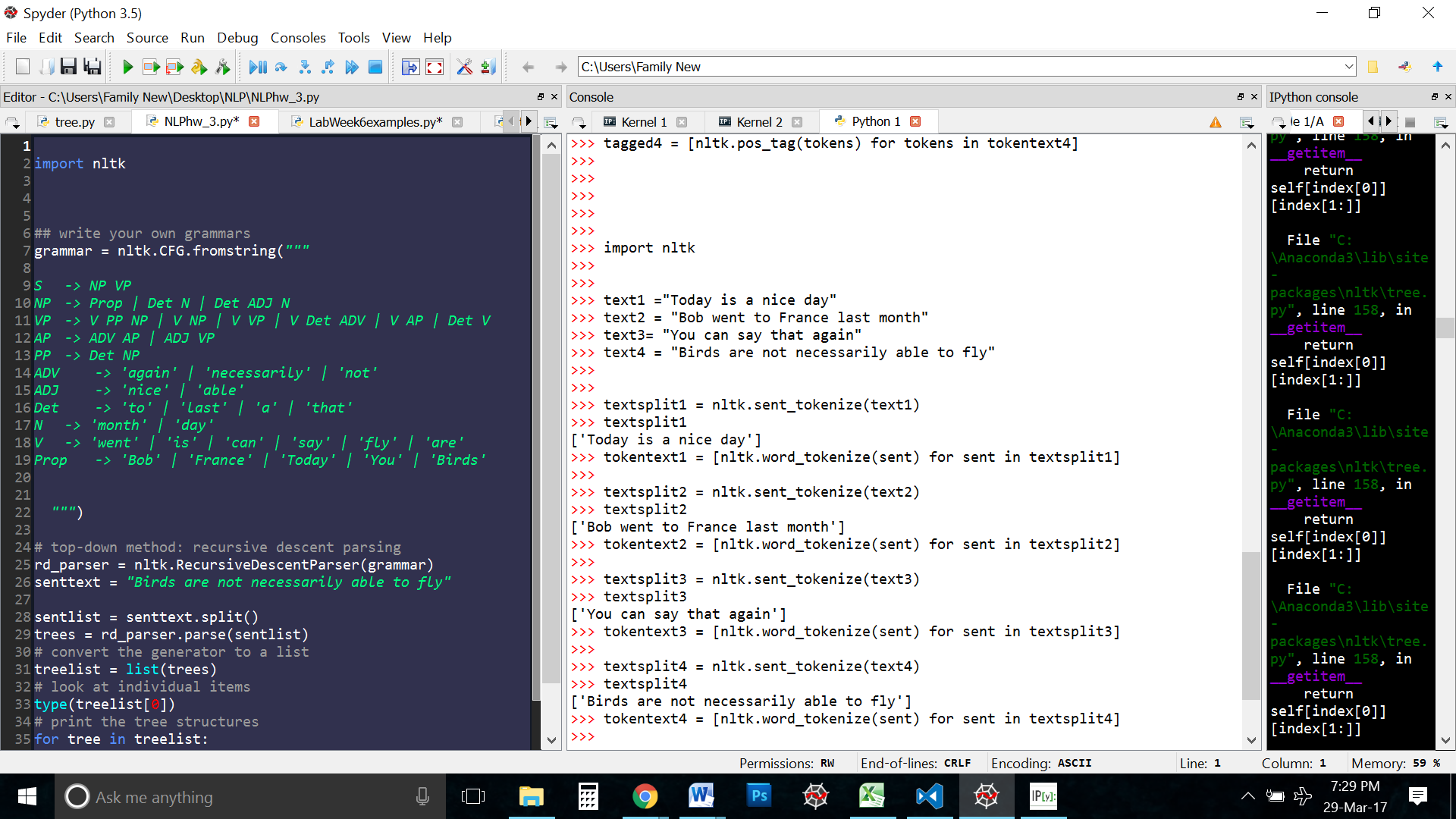
(b) “Bob went to France last month”

(c) “You can say that again”

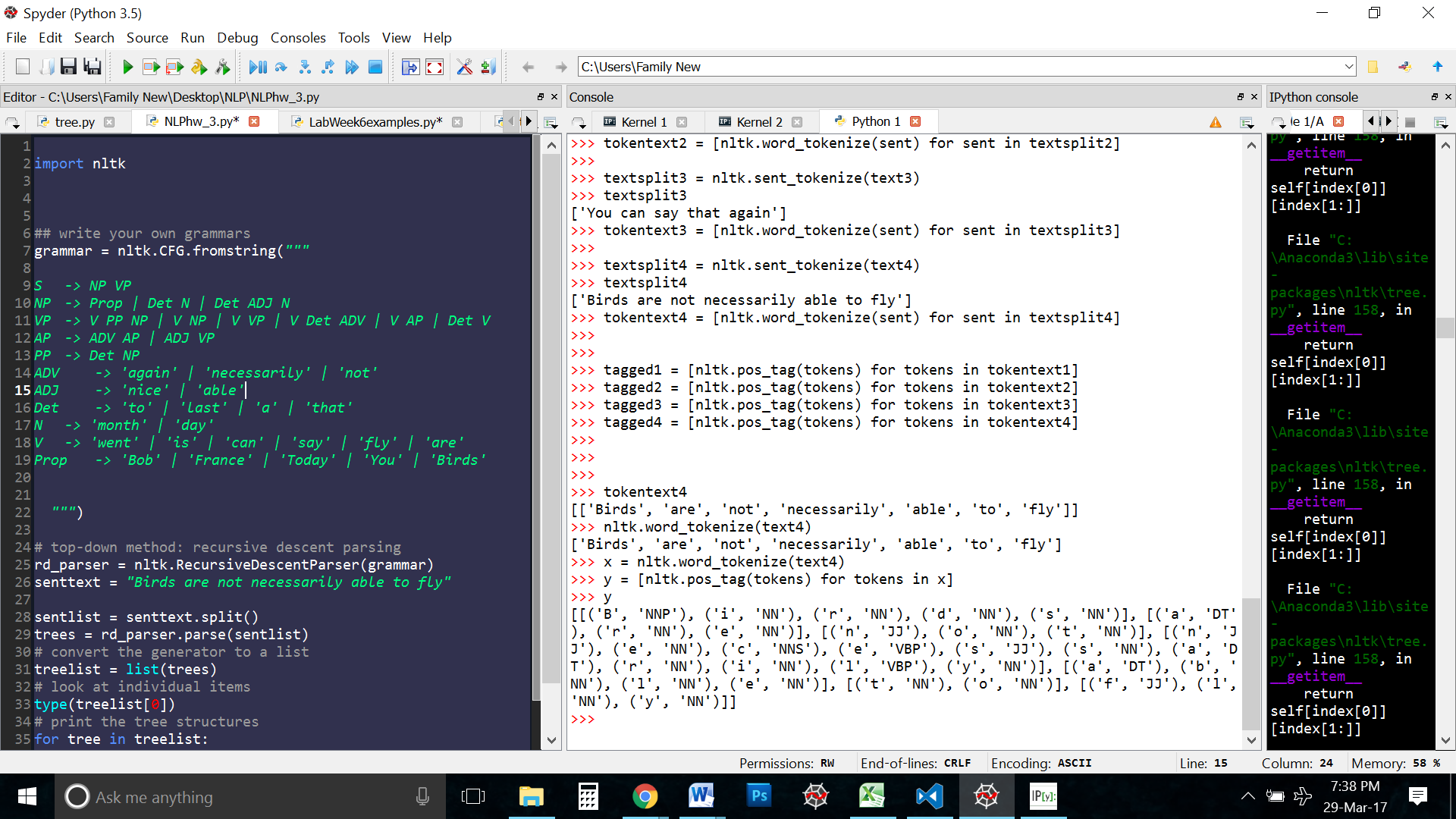
(d) “Birds are not necessarily able to fly”

This part proved to be challenging for me. UI was confused between the bottom-up versus the top-down approach. Ultimately in the end, because of the hint given by the professors, I decided to go with the bottom- up approach.

First, I imported NLTK into my python environment. Then I defined the 4 given sentences into variables. text1,text2…. respectively

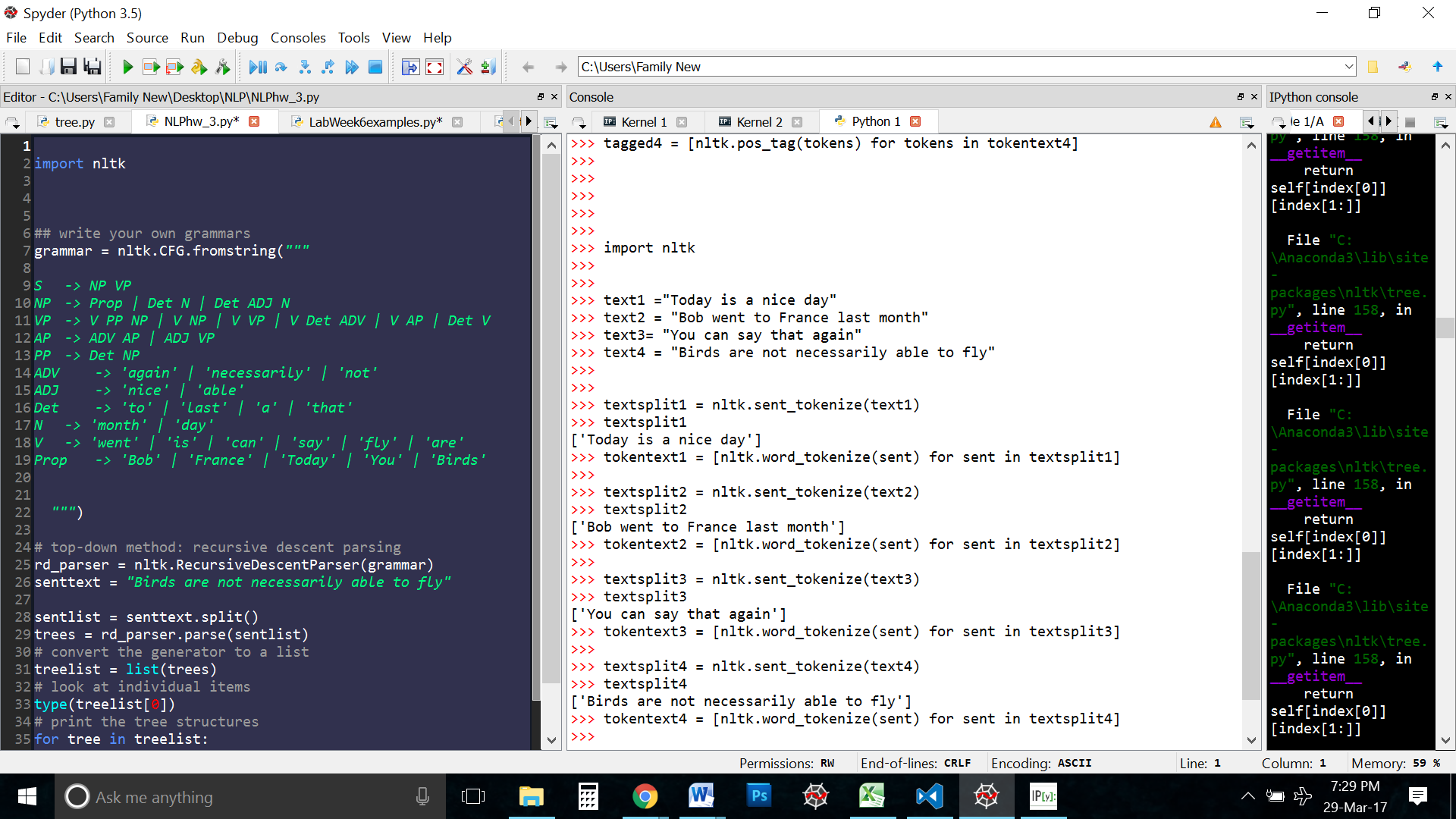


I first tokenized each sentence then further tokenized the words in the sentences. The sentences were tokenized twice because directly running the text through a word tokenizer then passing it through the POS tagger would result in something like this:



We do not want the individual alphabets. I don’t know why NLTK has them tagged in the corpus in the first place. What we need are the individual words. Hence we first tokenize the sentences the tokenize the words in that sentence.

After that, we run the sentences to the NLTK’s POS tagger to get the tags of the words

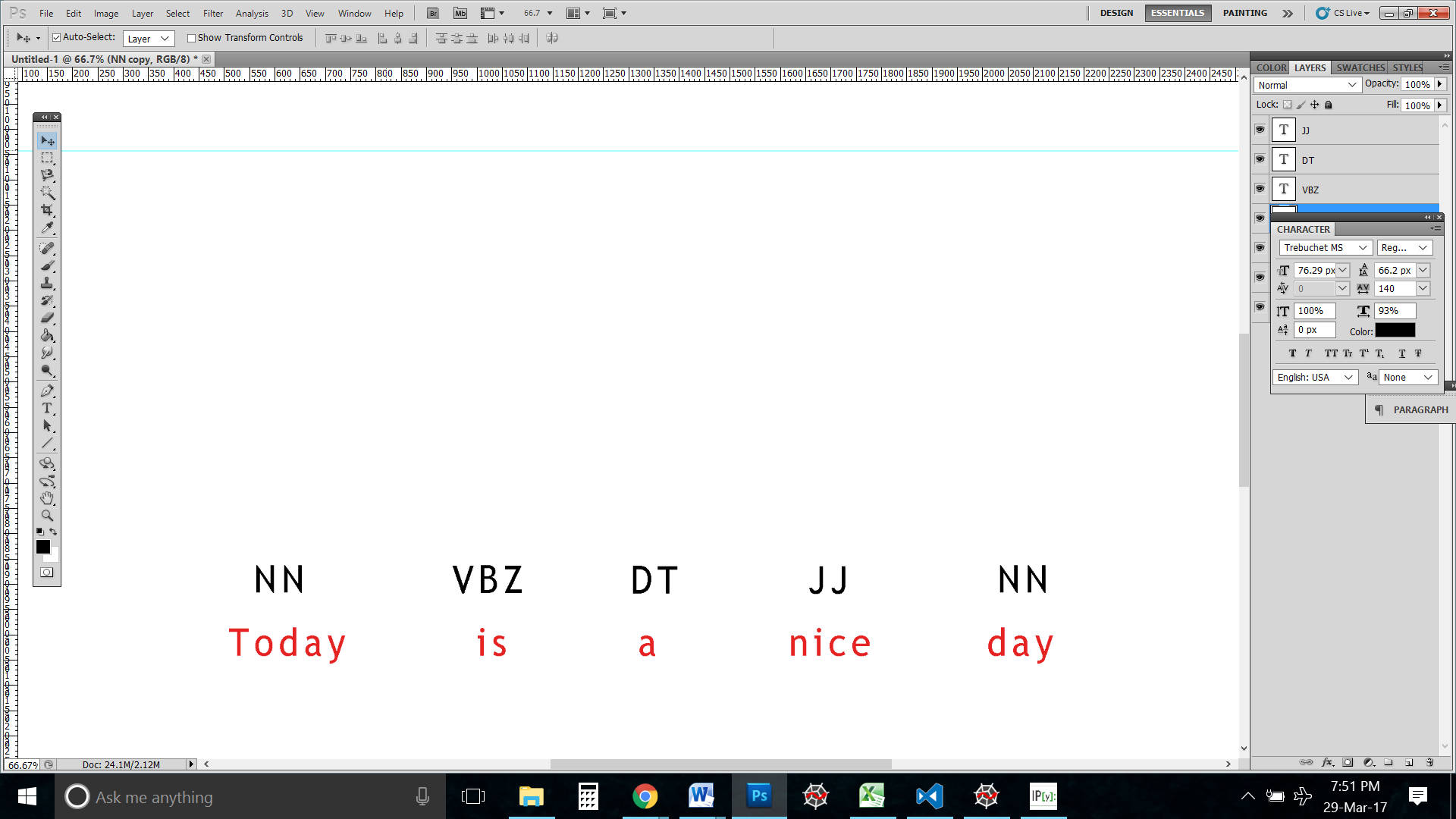


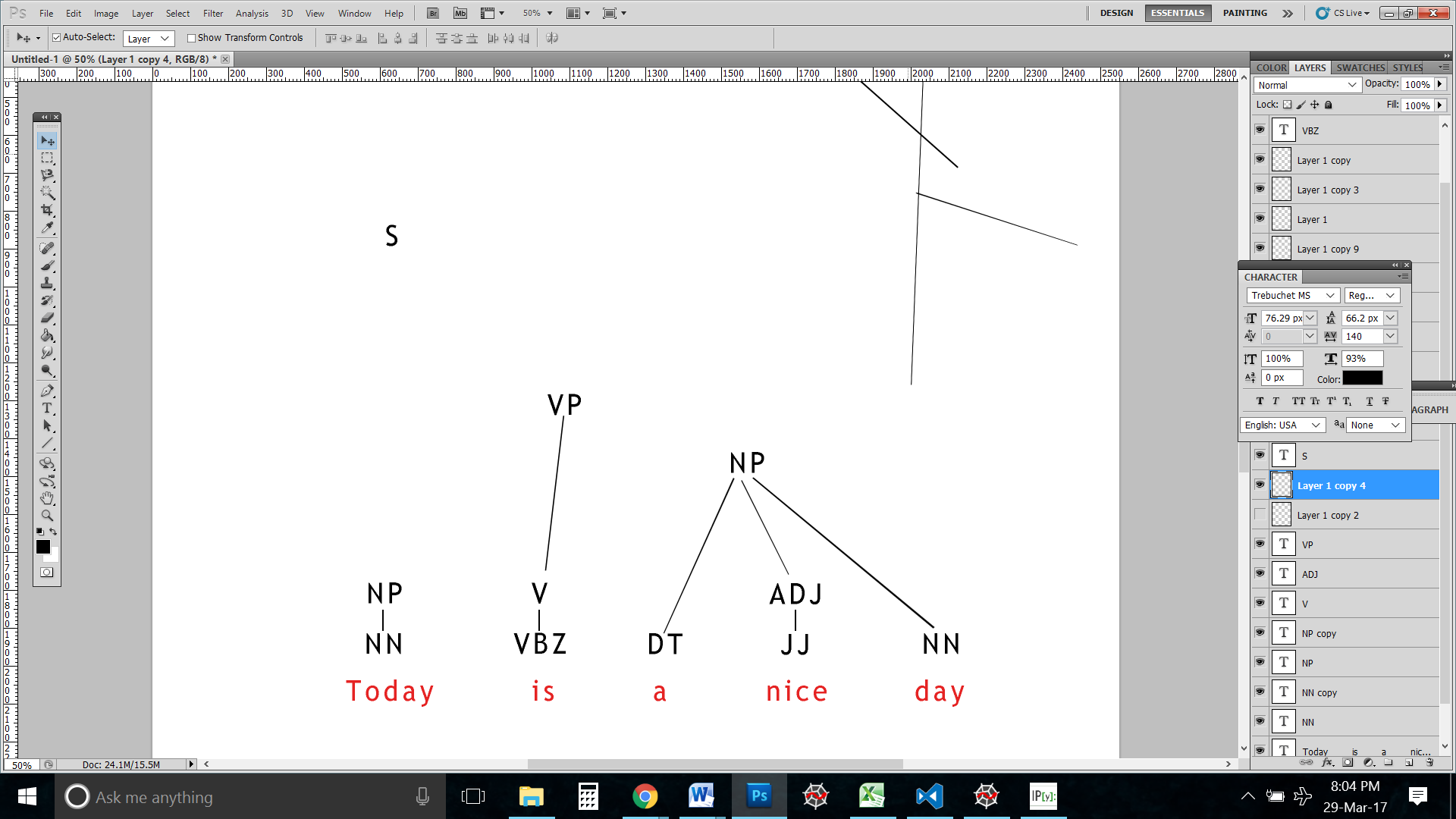
We get the following results

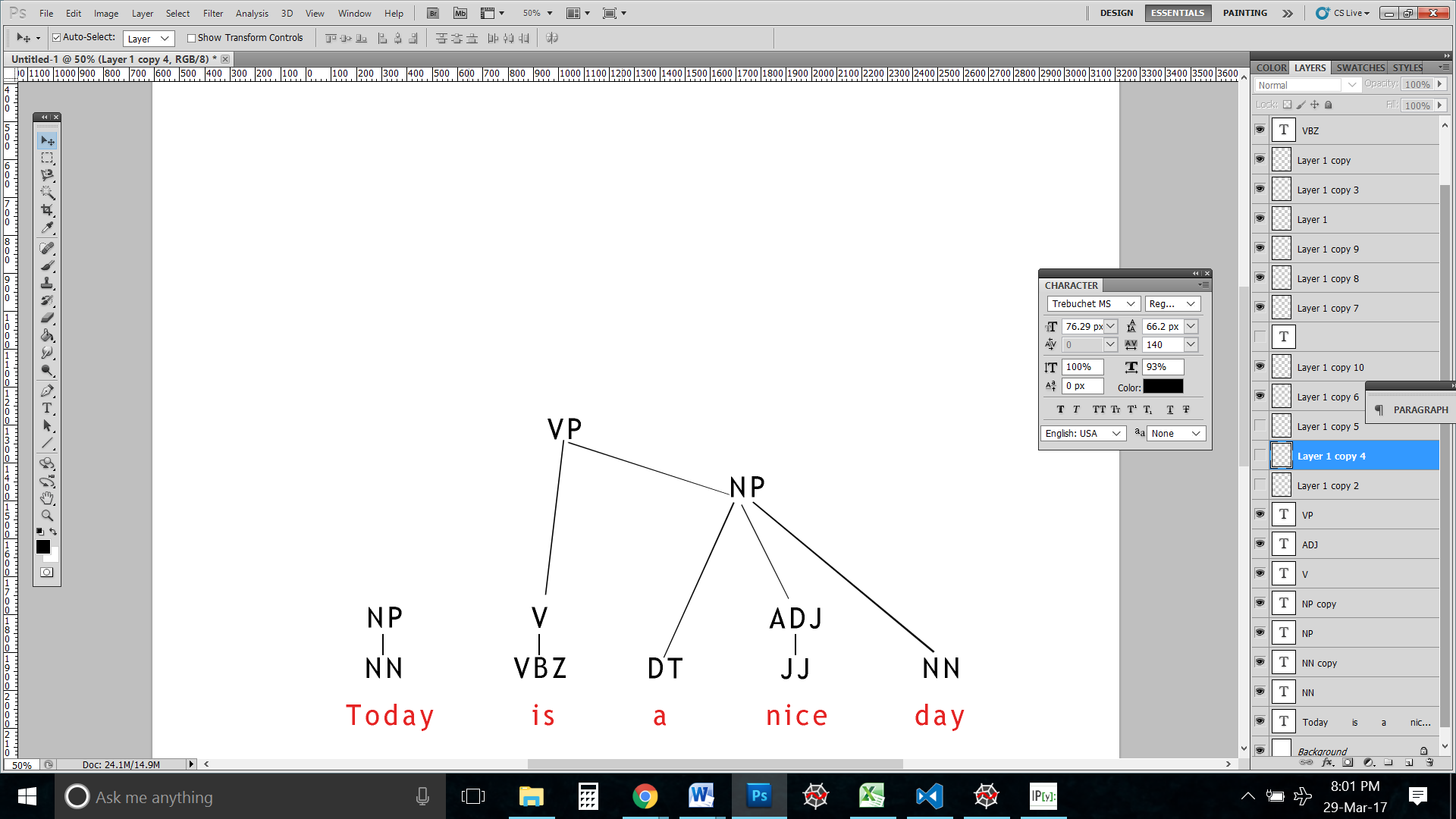


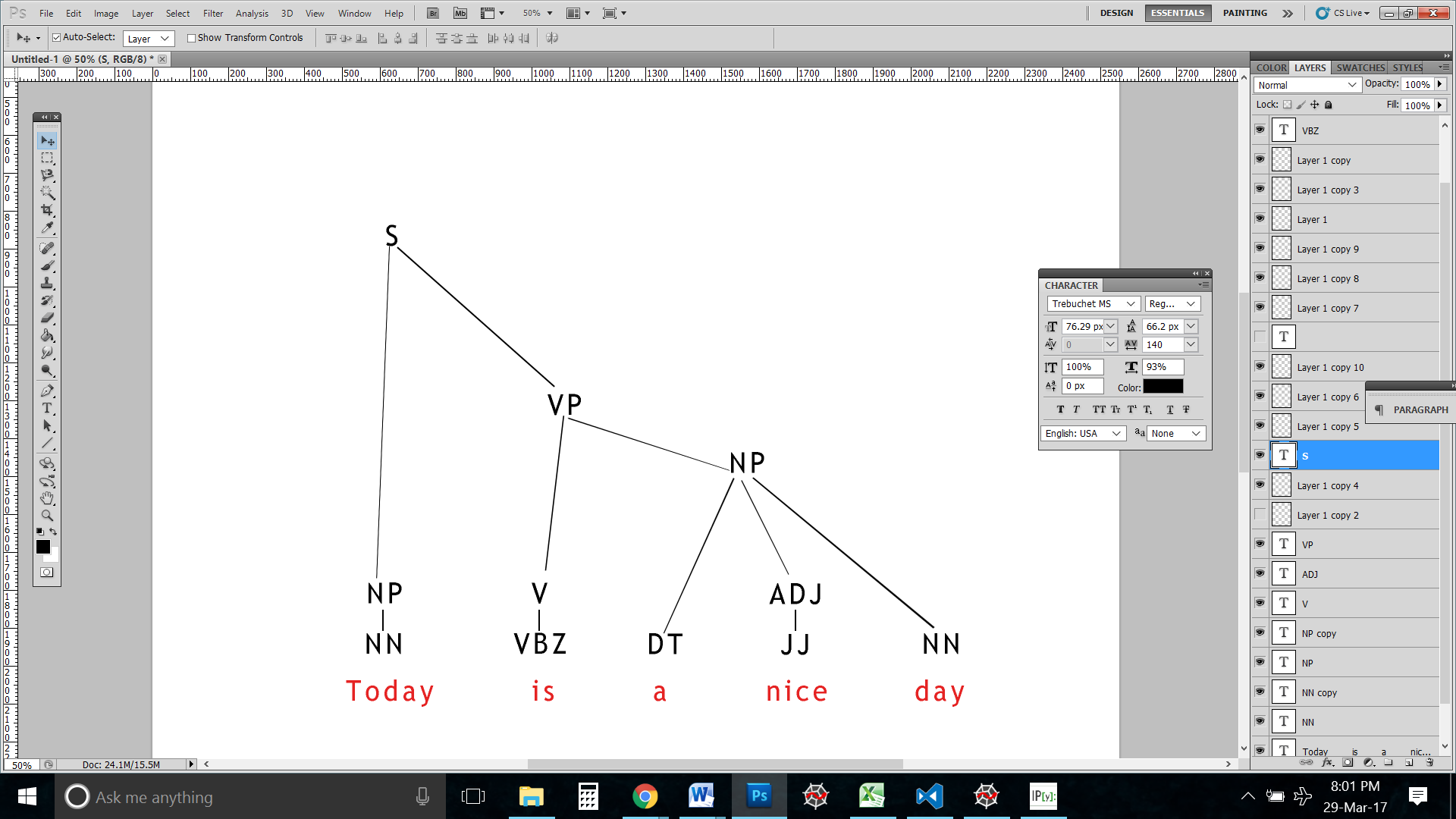
Then we proceed with the bottom up approach.

Here is an example for the sentence “Today is a nice day”









A similar process was used for all 4 sentences to come up with the following grammar:

**S -> NP VP**

**NP -> Prop | Det N | Det ADJ N**

**VP -> V PP NP | V NP | V VP | V Det ADV | V AP | Det V**

**AP -> ADV AP | ADJ VP**

**PP -> Det NP**

**ADV -> 'again' | 'necessarily' | 'not'**

**ADJ -> 'nice' | 'able'**

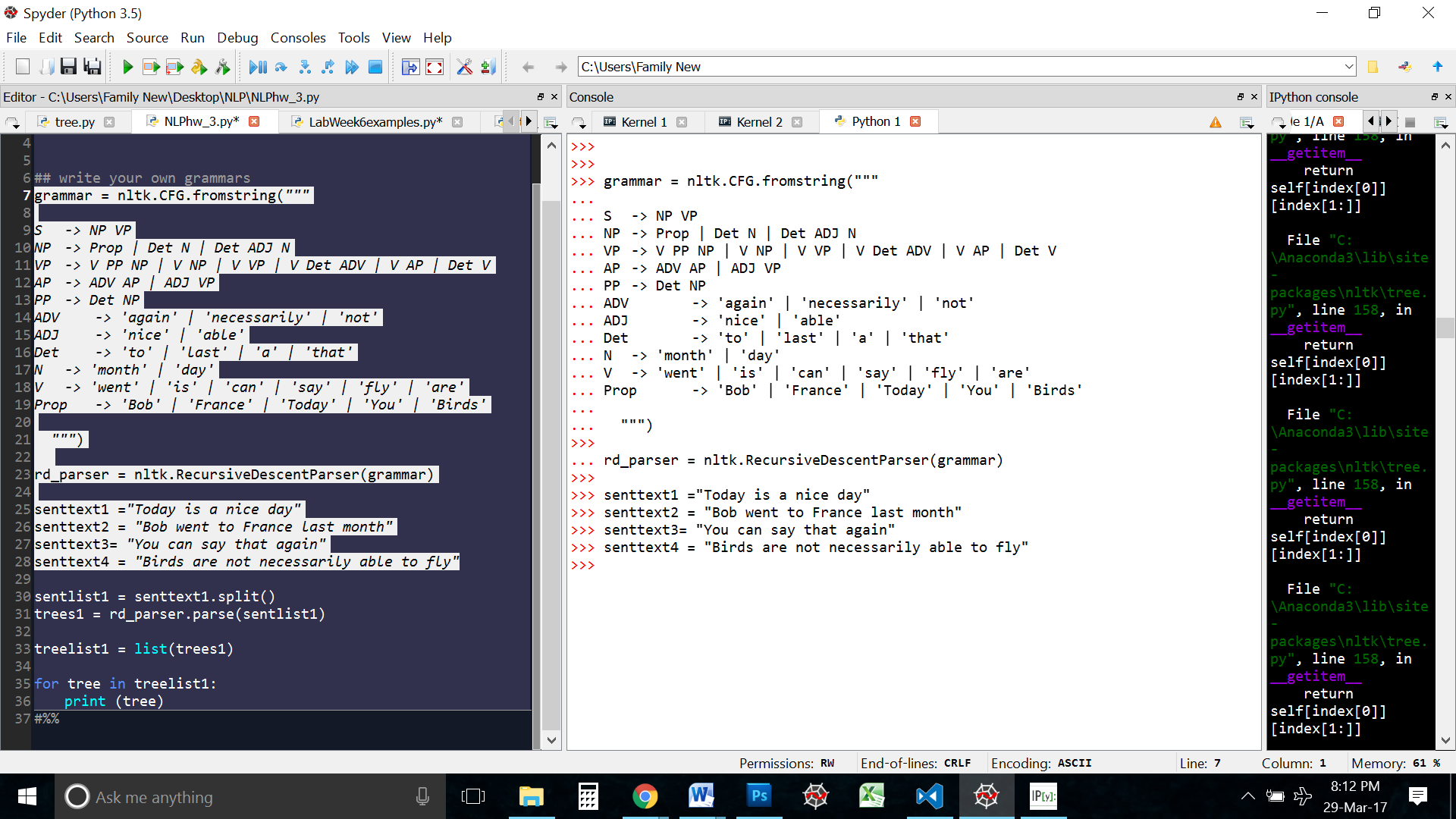
**Det -> 'to' | 'last' | 'a' | 'that'**

**N -> 'month' | 'day'**

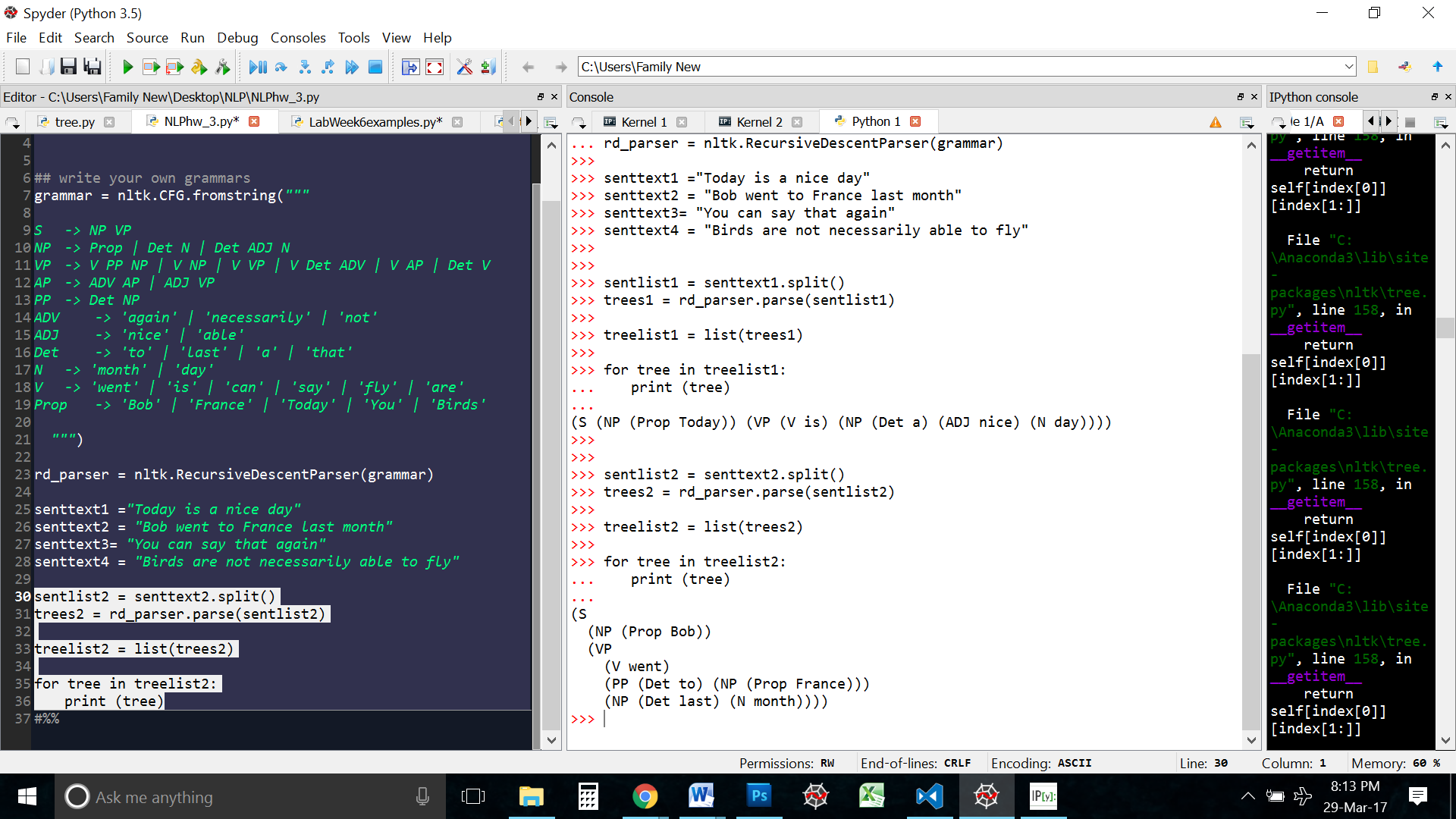
**V -> 'went' | 'is' | 'can' | 'say' | 'fly' | 'are'**

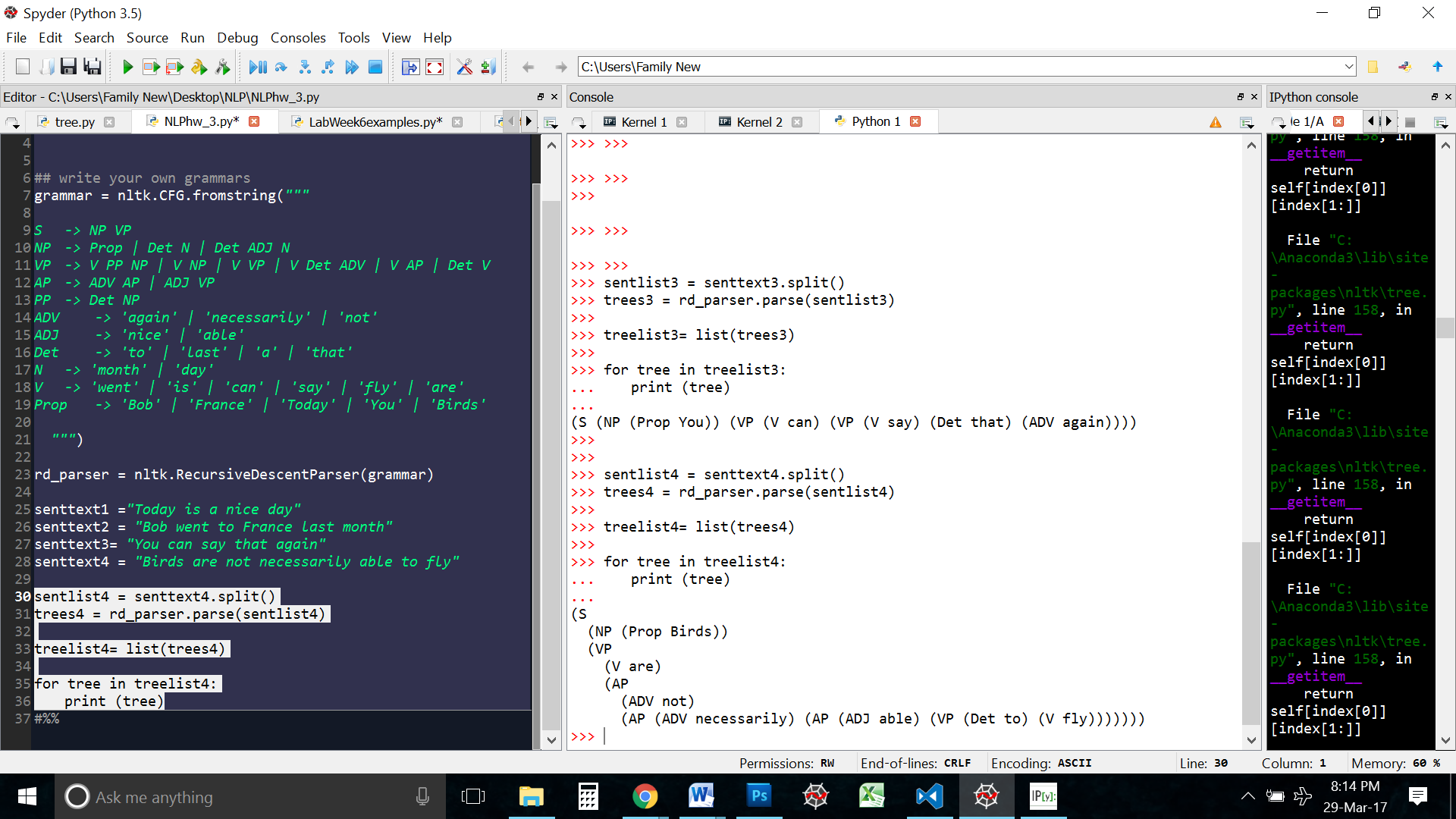
**Prop -> 'Bob' | 'France' | 'Today' | 'You' | 'Birds'**

**Note: While the approach to derive the grammar was bottom-up , I used the RecursiveDescentParser, which is a top-bottom parser.**



Then I parsed each of the sentences

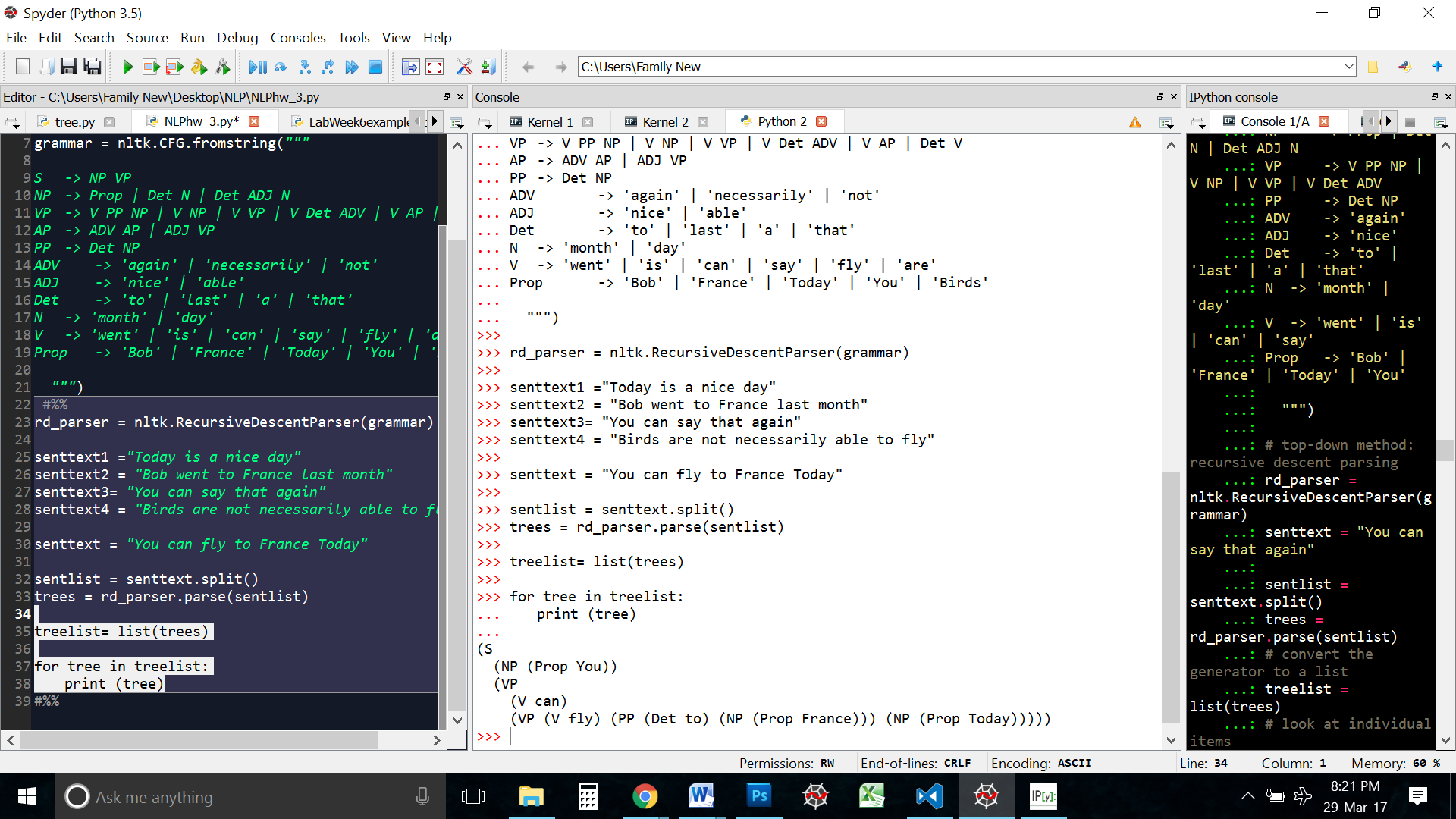




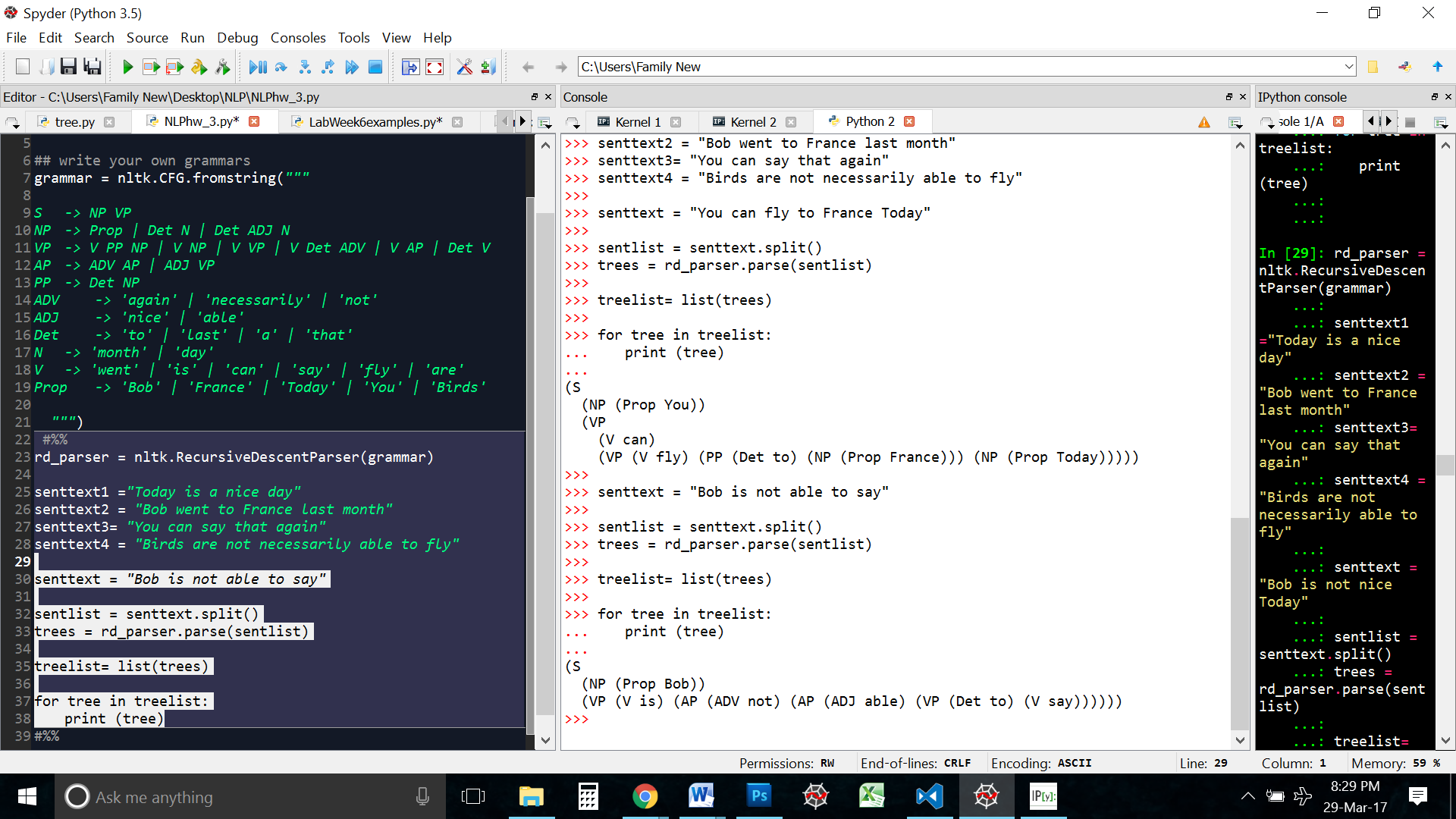
**--------------------------------------------------------------------------------**

1. **Three other sentences that can be passed by this grammar are:**

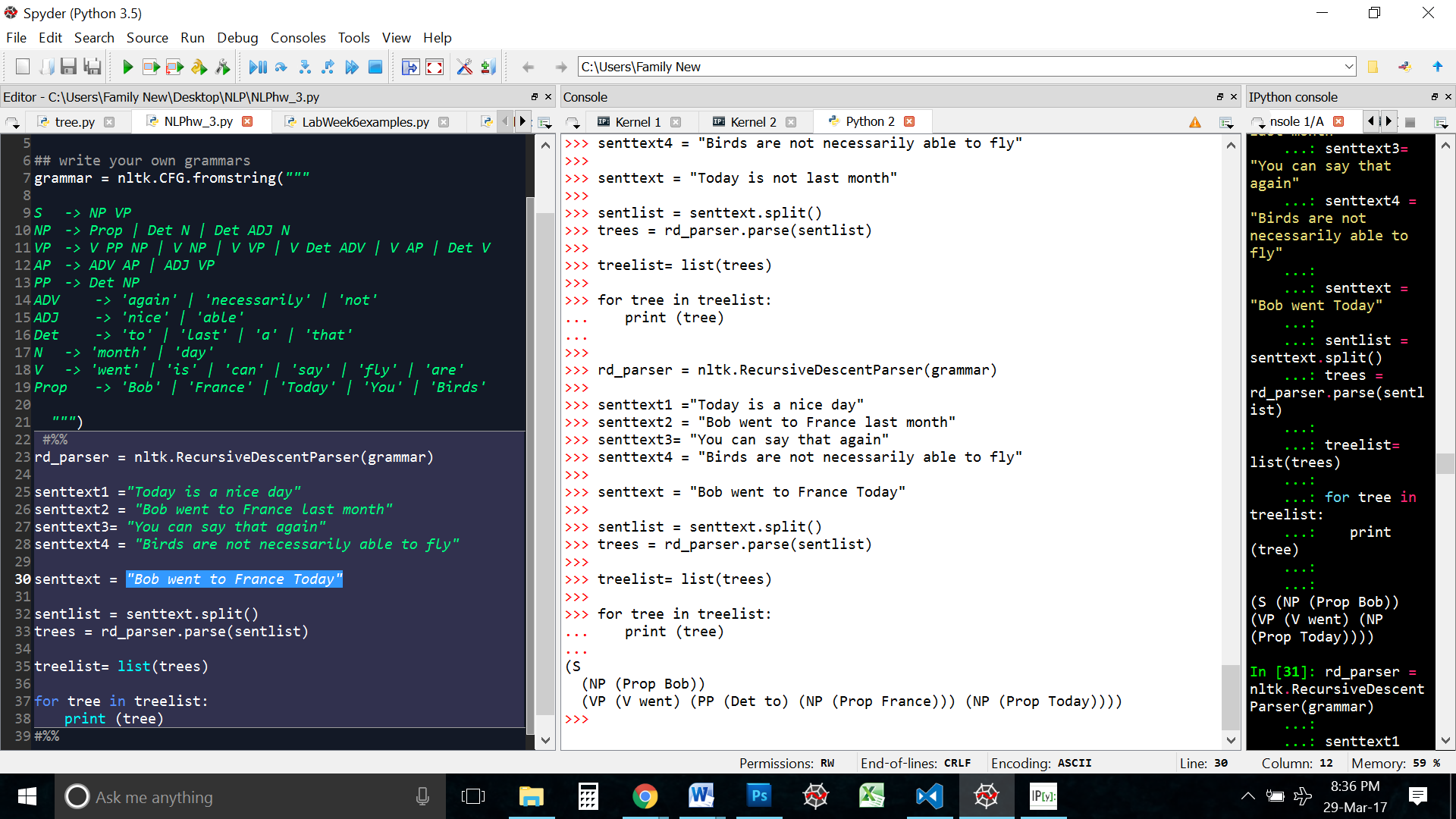
"You can fly to France Today"



"Bob is not able to say"



"Bob went to France Today"



This grammar is very limited. Because of that are only very few sentences that can be made. I tried making sentences that didn’t make sense, but they could not be passed by this grammar. I think one of the reasons is that this grammar is very strict. Like mentioned above there are very set rules and the terminal options are very restricted.

-------------------------------------------------------------------------------------------------------------------

1. **Probabilistic context free grammar.**

First I distributed the probability equally among each node. Then I parsed all the sentences

**S -> NP VP [1.0]**

**NP -> Prop[0.4] | Det N[0.3] | Det ADJ N[0.3]**

**VP -> V PP NP [0.16] | V NP [0.20] | V VP [0.16] | V Det ADV [0.16] | V AP [0.16] | Det V [0.16]**

**AP -> ADV AP[0.5] | ADJ VP [0.5]**

**PP -> Det NP [1.0]**

**ADV -> 'again' [0.3] | 'necessarily' [0.3] | 'not' [0.4]**

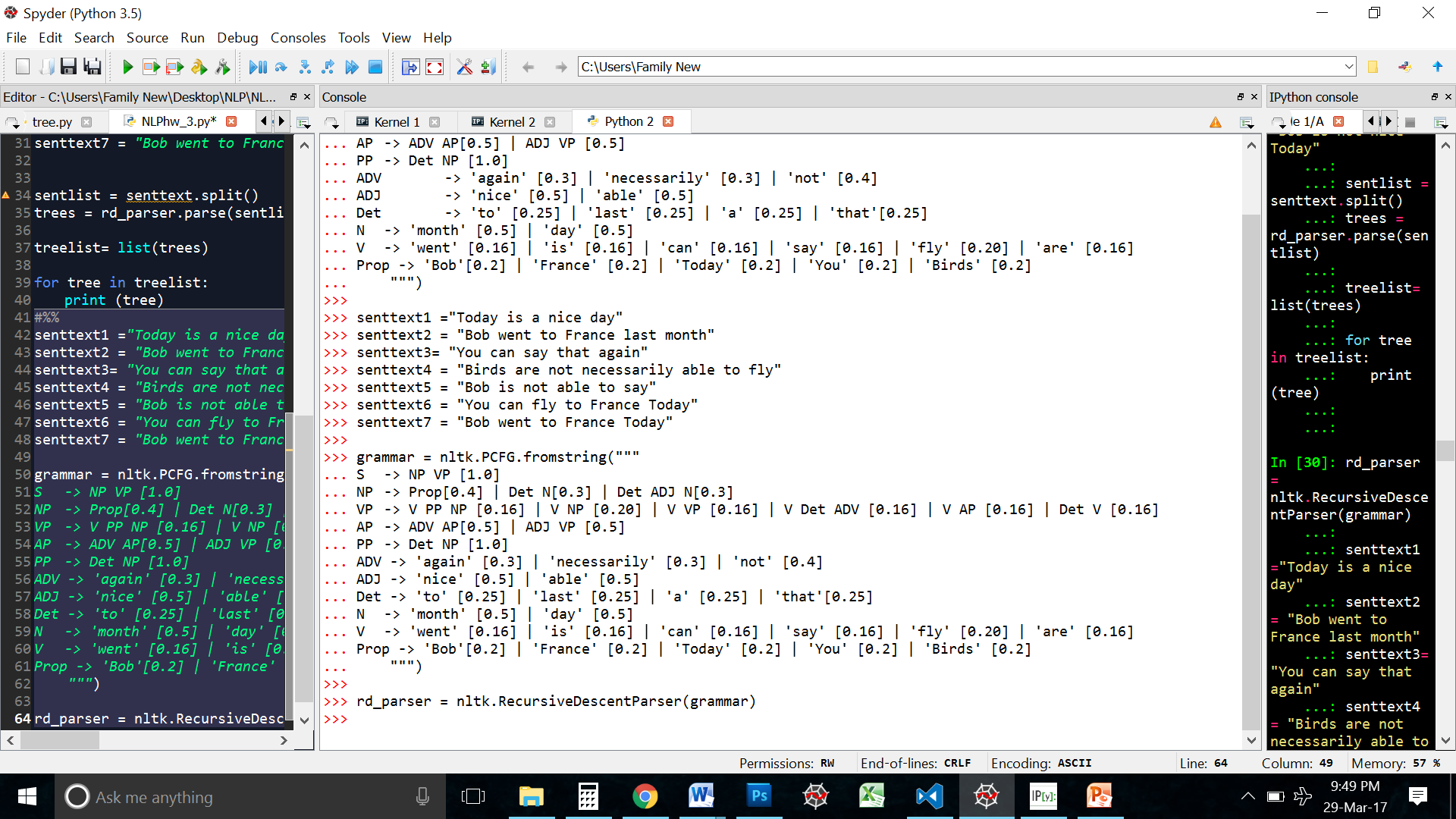
**ADJ -> 'nice' [0.5] | 'able' [0.5]**

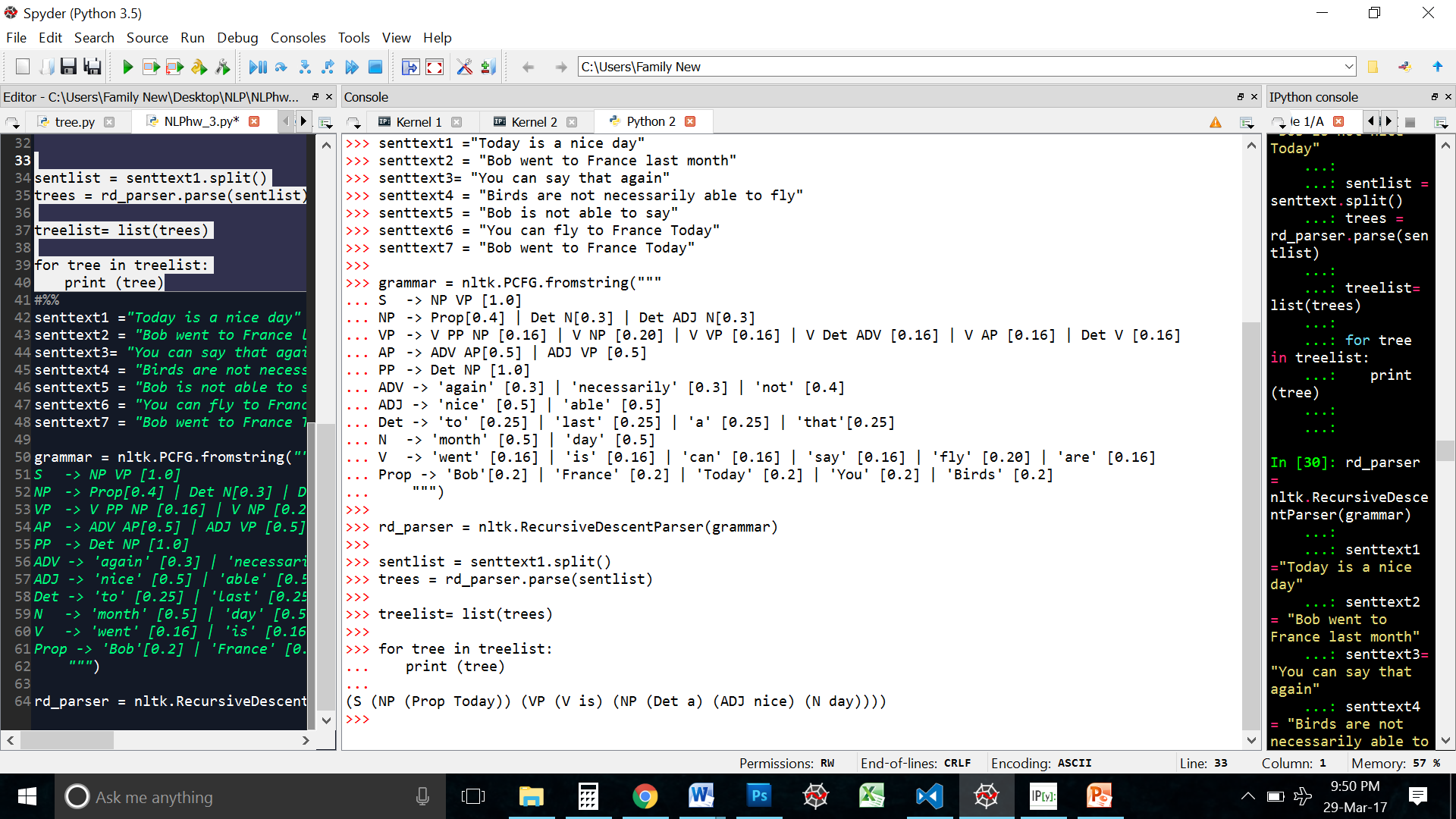
**Det -> 'to' [0.25] | 'last' [0.25] | 'a' [0.25] | 'that'[0.25]**

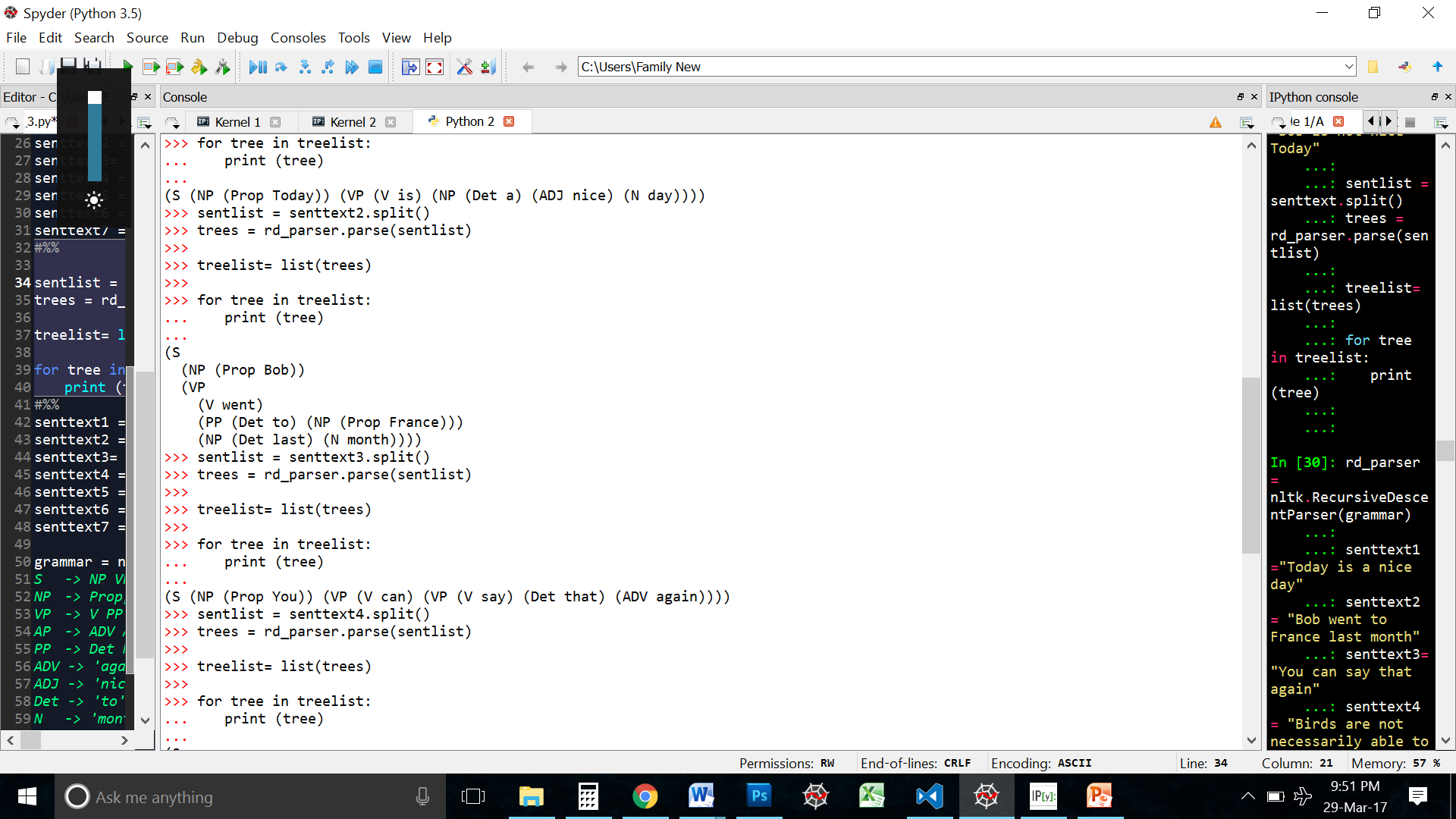
**N -> 'month' [0.5] | 'day' [0.5]**

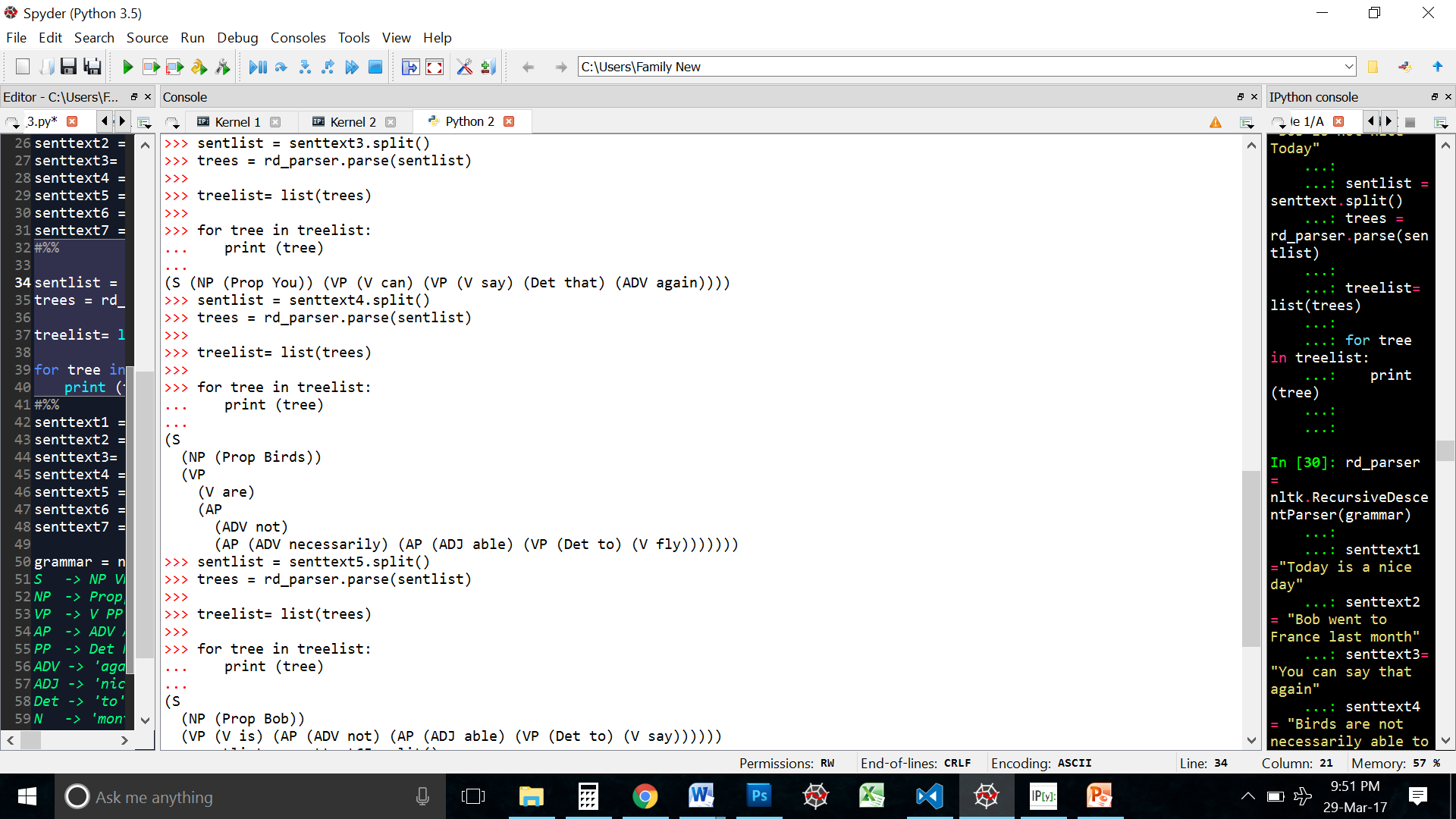
**V -> 'went' [0.16] | 'is' [0.16] | 'can' [0.16] | 'say' [0.16] | 'fly' [0.20] | 'are' [0.16]**

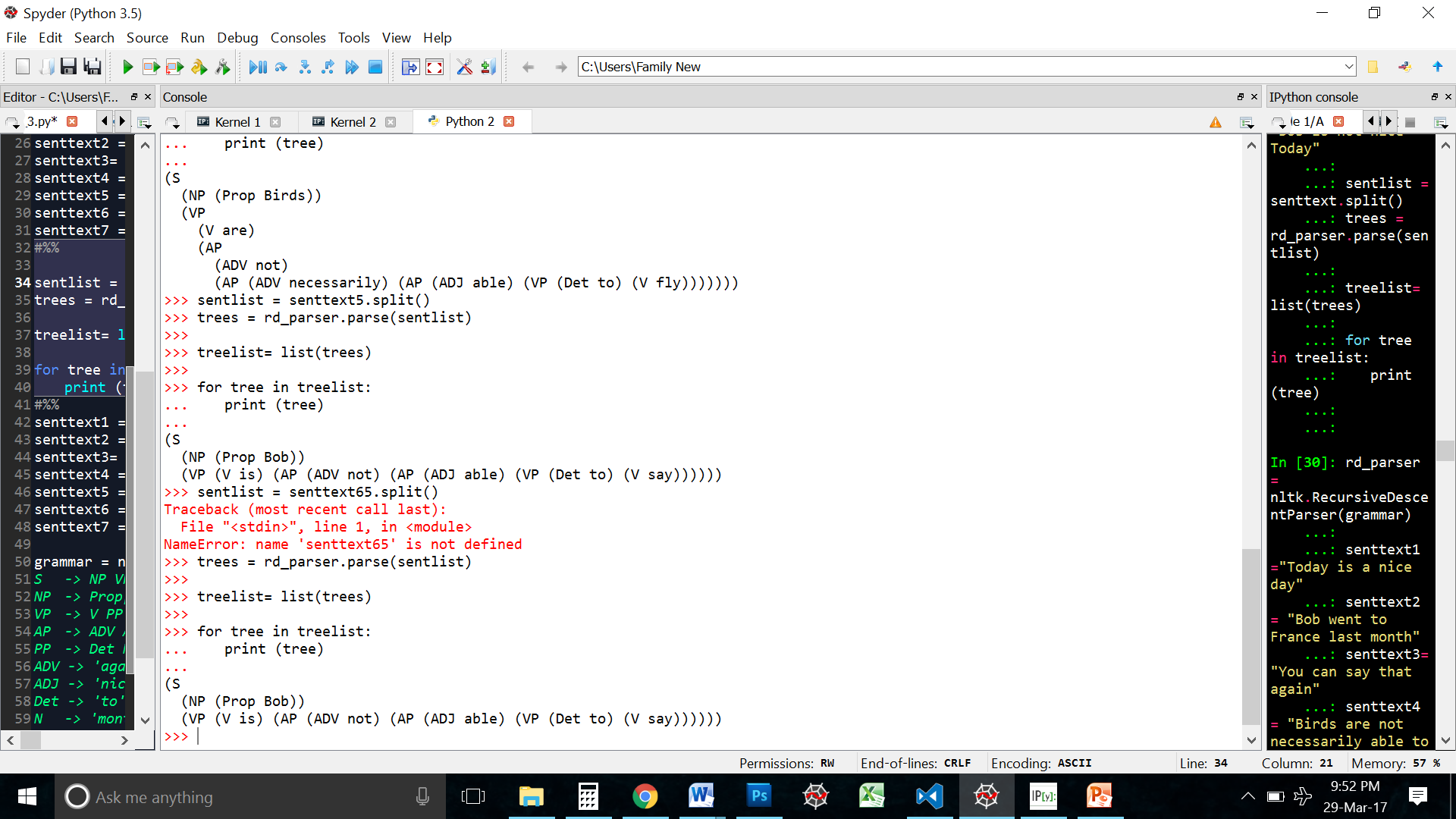
**Prop -> 'Bob'[0.2] | 'France' [0.2] | 'Today' [0.2] | 'You' [0.2] | 'Birds' [0.2]**

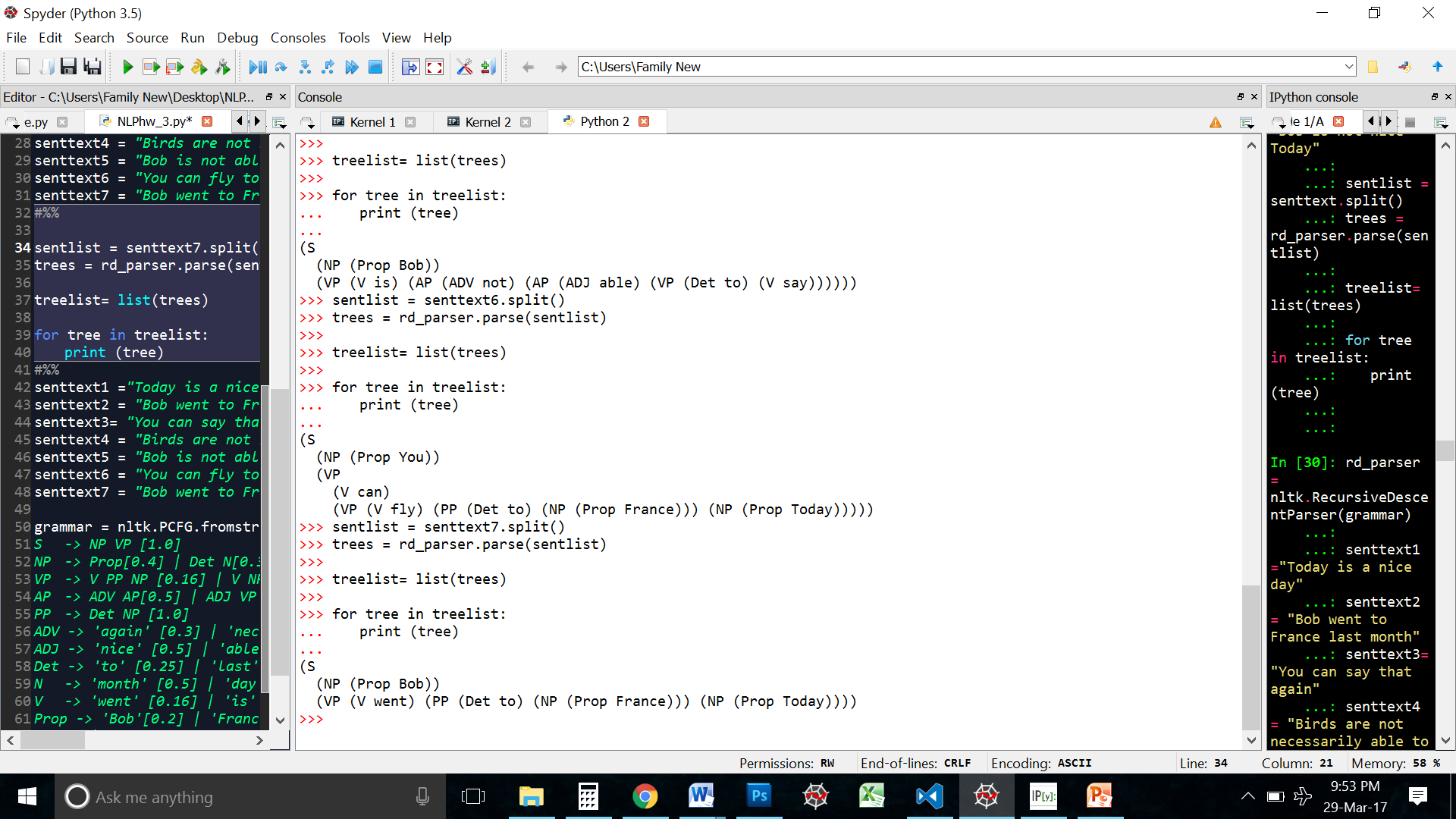












Since all the words were used once in the training set, I gave arbitrary probabilities to the terminals. The non-terminals, on the other hand, were a little more complex. VP -> V NP showed up in two sentences out of the four, while the other non-terminals (Except S -> NP VP) show up only once. Hence I gave more weight to VP -> V NP