The selected text for the assignment is an excerpt from Sam Altman's blog. Its URL is https://ia.samaltman.com/.

1. Code

```
import nltk
from collections import defaultdict
'''
excerpt = "First 8 sentences from the blog"
'''

tokens = nltk.word_tokenize(blogPost)
sentences = nltk.sent_tokenize(blogPost)

numberOftokens = 0
numberOfSentences = 0

for word in tokens:
    numberOftokens += 1

for sentence in sentences:
    numberOfSentences += 1

numberOftokensInSentenceTokenizer = 0

for sentence in sentences:
    tokensInSentence = nltk.word_tokenize(sentence)
    for word in tokensInSentence:
        numberOftokensInSentenceTokenizer += 1

lowerCaseTokens = [token.lower() for token in tokens] # list
   comprehension
stemmedTokensDict = {}

for algo in ["ORIGINAL_ALGORITHM",
   "MARTIN_EXTENSIONS","NLTK_EXTENSIONS"]:
    stemmer = nltk.PorterStemmer(mode=algo)
    stemmedTokens = [stemmer.stem(token) for token in lowerCaseTokens]
    stemmedTokensDict[algo] = stemmedTokens

for algo, tokens in stemmedTokensDict.items():
    numberOftokens = 0
    for token in tokens:
        numberOftokens += 1
    print("Number of tokens after stemming with ", algo, ":",
       numberOftokens)

posTags = nltk.pos_tag(tokens)
```

```
print("POS tags: ", posTags)
pos_tag_counts = defaultdict(int)
for word, tag in posTags:
    pos_tag_counts[tag] += 1
print("POS tags: ", pos_tag_counts)

bigrams = ngrams(tokens, 2)
trigrams = ngrams(tokens, 3)

def filter_ngrams(ngrams, min_count=2):
  filtered_ngrams = [(ngram,count) for ngram, count in
     Counter(ngrams).items() if count >= min_count]
  return filtered_ngrams

filtered_bigrams = filter_ngrams(bigrams)
filtered_trigrams = filter_ngrams(trigrams)

print("Filtered bigrams:")
for bigram, count in filtered_bigrams:
  print(bigram, count)

print("\nFiltered trigrams:")
for trigram, count in filtered_trigrams:
  print(trigram, count)
```

2.1. What differences did you notice between tokenizing by word versus by sentence?
The latter parses the text one sentence at a time, where one sentence is marked by the occurrence of '.', '!', and '?' characters as per the rules defined by English grammar. Building on this, it could be stated that the word tokenizer identifies various punctuation marks as individual tokens while only one sentence constitutes one token in the sentence tokenizer.

2.2. What POS tags were most frequent? Least frequent? Notice any errors?
Prepositions were most frequently used, however, 'TO', being a preposition, was not classified under that category. There isn't any tag for punctuation marks, though, they should be dropped from the result of this function. The least frequently used parts of speech were *'NNP'*, singular proper noun for the word **'AI'** and *'RP'* for 'out', which could again be classified as a preposition.

```
POS tags counter:  defaultdict(<class 'int'>, {'IN': 21, 'DT': 11,
   'JJ': 17, 'NN': 19, 'NNS': 16, ',': 5, 'PRP': 13, 'MD': 9, 'VB':
   15, 'TO': 7, 'WDT': 2, 'VBN': 7, 'PRP$': 5, '.': 8, 'VBZ': 2,
   'RB': 6, 'CC': 7, 'VBP': 7, 'RBR': 3, ':': 2, 'VBG': 2, 'CD': 1,
   'VBD': 2, 'NNP': 1, 'RP': 1}).
```

With the word 'TO', the current algorithm finds it difficult to differentiate between its use cases, for example, as an infinitive and a preposition. It is possibly because it doesn't take the context into account.

2.3. What do the most common bi-grams and tri-grams tell you about the content of the text?

```
Filtered bigrams:
('will', 'be') 3
('be', 'able') 2
('able', 'to') 2
('to', 'do') 2
('do', 'things') 2
('would', 'have') 2
(',', 'but') 2
('more', 'capable') 3
('we', 'can') 2
('benefit', 'from') 2
('that', 'we') 2
('  ', 't') 2

Filtered trigrams:
('will', 'be', 'able') 2
('be', 'able', 'to') 2
('able', 'to', 'do') 2
('to', 'do', 'things') 2
```

These commonly occurring ngrams where phrases and clauses in the text.

Here is the hyperlink to the notebook.