```
from collections import Counter
import re

text_corpus = "Unigrams are single words used in natural language processing. Unigrams are

tokens = re.findall(r'\b\w+\b', text_corpus.lower())

unigram_counts = Counter(tokens)

for unigram, count in unigram_counts.items():
    print(f'{unigram}: {count}')
```

```
unigrams: 2
are: 2
single: 1
words: 1
used: 1
in: 2
natural: 1
language: 1
processing: 1
often: 1
the: 1
first: 1
step: 1
text: 1
analysis: 1
```

```
import nltk
from nltk import bigrams
from nltk.tokenize import word_tokenize

text = "This is a sample text corpus for generating bigrams."
tokens = word_tokenize(text.lower())
bi_grams = list(bigrams(tokens))
print("Bigrams:", bi_grams)
```

```
Bigrams: [('this', 'is'), ('is', 'a'), ('a', 'sample'), ('sample', 'text'), ('t
```

```
import nltk
nltk.download('punkt')

from nltk import word_tokenize, ngrams
from collections import Counter, defaultdict
import random
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
corpus = "Tokenization is the process of splitting text into individual words or phrases."
tokens = word_tokenize(corpus.lower())
unigrams = tokens
print("Unigrams:", unigrams)
bigrams = list(ngrams(tokens, 2))
```

```
print("Bigrams:", bigrams)
trigrams = list(ngrams(tokens, 3))
print("Trigrams:", trigrams)
```

```
Unigrams: ['tokenization', 'is', 'the', 'process', 'of', 'splitting', 'text', '
Bigrams: [('tokenization', 'is'), ('is', 'the'), ('the', 'process'), ('process'
Trigrams: [('tokenization', 'is', 'the'), ('is', 'the', 'process'), ('the', 'pr
```

```
bigram_counts = Counter(bigrams)
unigram_counts = Counter(unigrams)
bigram_probabilities = {bigram: count / unigram_counts[bigram[0]] for bigram, count in big
print("Bigram Probabilities:", bigram_probabilities)
```

```
Bigram Probabilities: {('tokenization', 'is'): 1.0, ('is', 'the'): 1.0, ('the',
```

```
def predict_next_word(prev_word):
   possible_bigrams = {bigram: prob for bigram, prob in bigram_probabilities.items() if bi
   if not possible_bigrams:
      return None
   next_word = max(possible_bigrams, key=possible_bigrams.get)[1]
   return next_word


previous_word = "bigrams"
predicted_word = predict_next_word(previous_word)
print(f"Next word prediction for '{previous_word}': {predicted_word}")
```

```
Next word prediction for 'bigrams': None
```

```
import nltk
from nltk import bigrams
from nltk.probability import FreqDist
from nltk.tokenize import word_tokenize

text_corpus = """
Natural language processing (NLP) is a sub-field of artificial intelligence (AI) that focu
"""

tokens = word_tokenize(text_corpus.lower())

bigram_list = list(bigrams(tokens))

bigram_freq = FreqDist(bigram_list)
word_freq = FreqDist(tokens)

bigram_probabilities = {}
for (w1, w2), count in bigram_freq.items():
    prob = count / word_freq[w1]
    bigram_probabilities[(w1, w2)] = prob

print("Bigram Probabilities:")
for bigram, prob in bigram_probabilities.items():
    print(f"P({bigram[1]} | {bigram[0]}) = {prob:.6f}")
```

```
Bigram Probabilities:
P(language | natural) = 1.000000
P(processing | language) = 0.333333
```

P((| processing) = 1.000000
P(nlp | () = 0.500000
P() | nlp) = 0.333333
P(is |)) = 0.500000
P(a | is) = 0.333333
P(sub-field | a) = 0.500000
P(of | sub-field) = 1.000000
P(artificial | of) = 0.333333
P(intelligence | artificial) = 1.000000
P((| intelligence) = 1.000000
P(ai | () = 0.500000
P() | ai) = 1.000000
P(that |)) = 0.500000
P(focuses | that) = 0.500000
P(on | focuses) = 1.000000
P(the | on) = 1.000000
P(interaction | the) = 0.500000
P(between | interaction) = 1.000000
P(computers | between) = 1.000000
P(and | computers) = 0.500000
P(humans | and) = 0.250000
P(through | humans) = 1.000000
P(natural | through) = 1.000000
P(. | language) = 0.333333
P(the | .) = 0.333333
P(ultimate | the) = 0.500000
P(objective | ultimate) = 1.000000
P(of | objective) = 1.000000
P(nlp | of) = 0.666667
P(is | nlp) = 0.333333
P(to | is) = 0.333333
P(enable | to) = 0.500000
P(computers | enable) = 1.000000
P(to | computers) = 0.500000
P(understand | to) = 0.500000
P(, | understand) = 1.000000
P(interpret | ,) = 0.250000
P(, | interpret) = 1.000000
P(and | ,) = 0.500000
P(generate | and) = 0.250000
P(human | generate) = 1.000000
P(language | human) = 1.000000
P(in | language) = 0.333333
P(a | in) = 1.000000
P(way | a) = 0.500000
P(that | way) = 1.000000
P(is | that) = 0.500000
P(both | is) = 0.333333
P(valuable | both) = 1.000000
P(and | valuable) = 1.000000
P(meaningful | and) = 0.250000
P(. | meaningful) = 1.000000
P(key | .) = 0.333333
P(applications | key) = 1.000000
P(of | applications) = 1.000000