```
import random
from collections import defaultdict, Counter

# Sample text
text = "This is a simple example of a bigram model. The bigram model generates text based

# Tokenize the text
tokens = text.lower().split()

# Create bigram model
bigrams = list(zip(tokens, tokens[1:]))
bigram_model = defaultdict(Counter)

for w1, w2 in bigrams:
    bigram_model[w1][w2] += 1

# Function to generate text
def generate_text(start_word, num_words=10):
    current_word = start_word
    output = [current_word]

    for _ in range(num_words - 1):
        next_words = bigram_model[current_word]
        if not next_words:
            break
        current_word = random.choices(list(next_words.keys()), list(next_words.values()))
        output.append(current_word)

    return ' '.join(output)

# Generate text
print(generate_text('the', num_words=10))
```

```
the previous word.
```

```
import nltk
from nltk.tokenize import word_tokenize

# Download required NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Sample text
text = "This is a simple example of part-of-speech tagging using NLTK."

# Tokenize and tag
tokens = word_tokenize(text)
tagged = nltk.pos_tag(tokens)

print(tagged)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('simple', 'JJ'), ('example', 'NN'), ('c
```

```
from collections import defaultdict, Counter
import random

# Sample corpus (for demonstration purposes)
corpus = [
    ("This", "DT"), ("is", "VBZ"), ("a", "DT"), ("simple", "JJ"), ("example", "NN"),
    ("of", "IN"), ("part-of-speech", "NN"), ("tagging", "NN"), ("using", "VBG"),
    ("NLTK", "NNP")
]

# Create frequency distributions
word_tag_counts = defaultdict(Counter)
tag_counts = Counter()

for word, tag in corpus:
    word_tag_counts[word][tag] += 1
    tag_counts[tag] += 1

# Function to assign POS tags
def stochastic_tagging(word):
    if word in word_tag_counts:
        return random.choices(list(word_tag_counts[word].keys()), list(word_tag_counts[wo
    else:
        return random.choice(list(tag_counts.keys()))

# Test tagging
words = ["This", "is", "a", "tagging", "example"]
tags = [(word, stochastic_tagging(word)) for word in words]
print(tags)
```

```
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('tagging', 'NN'), ('example', 'NN')]
```

```
import re

def rule_based_tagging(word):
    if re.match(r'^[A-Z][a-z]*$', word):
        return 'NNP'  # Proper noun
    elif re.match(r'^[a-z]+$', word):
        return 'NN'  # Noun
    elif re.match(r'^is|am|are$', word):
        return 'VB'  # Verb
    return 'NN'

# Test tagging
words = ["This", "is", "a", "test", "example"]
tags = [(word, rule_based_tagging(word)) for word in words]
print(tags)
```

```
[('This', 'NNP'), ('is', 'NN'), ('a', 'NN'), ('test', 'NN'), ('example', 'NN')]
```

```
def initial_tagging(word):
    if word.lower() in ["is", "am", "are"]:
        return 'VB'
    elif word.lower() in ["the", "a"]:
        return 'DT'
    else:
        return 'NN'

def apply_rules(tagged_words):
    # Basic rule example: convert NN to NNS if followed by plural noun
    new_tags = []
    for i, (word, tag) in enumerate(tagged_words):
        if tag == 'NN' and i + 1 < len(tagged_words) and tagged_words[i + 1][1] ==
            new_tags.append((word, 'NNS'))
        else:
            new_tags.append((word, tag))
    return new_tags

# Test transformation-based tagging
words = ["The", "cats", "are", "on", "the", "rooftop"]
initial_tags = [(word, initial_tagging(word)) for word in words]
transformed_tags = apply_rules(initial_tags)
print(transformed_tags)
```

```
[('The', 'DT'), ('cats', 'NN'), ('are', 'VB'), ('on', 'NN'), ('the', 'DT'), ('rooftop
```