```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg
from collections import defaultdict, Counter
import random
```

```
nltk.download('gutenberg')
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
True
```

```
corpus = gutenberg.raw("/content/1661-0.txt")
```

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
tokens = word_tokenize(corpus)
```

```
freq_dict = defaultdict(Counter)
```

```
for i in range(len(tokens) - 1):
    word, next_word = tokens[i], tokens[i + 1]
    freq_dict[word][next_word] += 1
```

```
prob_dict = defaultdict(dict)
for word, next_words in freq_dict.items():
    total = sum(next_words.values())
    for next_word, count in next_words.items():
        prob_dict[word][next_word] = count / total
```

```
def generate_next_word(word):
    next_words = prob_dict[word]
    if len(next_words) == 0:
        return None
    return random.choices(list(next_words.keys()), list(next_words.values()))[0]
```

```
def generate_sentence(start_word, length=100):
    sentence = [start_word]
    for i in range(length):
        next_word = generate_next_word(sentence[-1])
        if next_word is None:
            break
        sentence.append(next_word)
    return ' '.join(sentence)
```

```
print(generate_sentence('In', length=1000))
```

```
In life than the features and what I should possess so much astonished , thereâ s always as usual signal I deduced from death of h
```

In this code, we first load the data from the Gutenberg corpus and tokenize it using the word_tokenize function

from the NLTK library. We then create a dictionary to store the frequency of each word using the defaultdict and Counter

classes from the Python collections module.

Next, we generate the probability of the next word given the current word using the frequency dictionary.

We define a function generate_next_word to generate the next word given the current word based on the probability dictionary.

Finally, we define a function generate_sentence to generate a sentence by randomly selecting the next word based on the probability of the next word given the current word. We generate a sentence by calling the generate_sentence function with a starting word and the length of the sentence.