Experiment No:-7. Write a program in python to implement a bi-gram, tri-gram language model to suggest the next possible words in a sentence based on brown word corpus and its probability of selection of every word.Extending this, complete entire sentences.

```
import nltk
from nltk.corpus import brown
from collections import Counter, defaultdict
import random

# Download Brown corpus if not already available

nltk.download('brown')
nltk.download('punkt')

# Tokenize the Brown corpus
brown_words = list(brown.words())

# Function to build n-gram model
def build_ngram_model(words, n=2):
    ngram_counts = defaultdict(Counter)
    total_counts = Counter()

    for i in range(len(words) - n + 1):
        prefix = tuple(words[i:i+n-1])
        next_word = words[i+n-1]
        ngram_counts[prefix][next_word] += 1
        total_counts[prefix] += 1

    # Convert counts to probabilities
    ngram_model = {prefix: {word: count / total_counts[prefix] for word, count in suffix.items()} for prefix, suffix in ngram_counts.item

    return ngram_model

# Build bi-gram and tri-gram models
bigram_model = build_ngram_model(brown_words, 2)
trigram_model = build_ngram_model(brown_words, 3)

# Function to suggest next words
def suggest_next_word(prefix, model, top_n=5):
    prefix = tuple(prefix)
    if prefix in model:
        return sorted(model[prefix].items(), key=lambda x: x[1], reverse=True)[:top_n]
    return []

# Function to generate a sentence based on n-gram model
def generate_sentence(start_words, model, max_length=15):
    sentence = list(start_words)

    for _ in range(max_length - len(start_words)):
        suggestions = suggest_next_word(sentence[-(len(start_words)):], model)
        if not suggestions:
            break
        next_word = random.choices([word for word, _ in suggestions], weights=[prob for _, prob in suggestions])[0]
        sentence.append(next_word)
        if next_word in ['.', '!', '?']:
            break

    return ' '.join(sentence)



start_words = ["The"]
print("Bi-gram suggestions:", suggest_next_word(start_words, bigram_model))
print("Tri-gram suggestions:", suggest_next_word(["The", "United"], trigram_model))
print("Generated sentence:", generate_sentence(start_words, bigram_model))
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Bi-gram suggestions: [('first', 0.013226784238082117), ('only', 0.007577845136401212), ('man', 0.006613392119041058), ('new', 0.005
Tri-gram suggestions: [('States', 0.8888888888888888), ('Nations', 0.1111111111111111)]
Generated sentence: The man in a little to the first , and a man , the most
```