```
import nltk              # for tokenization (sentences & words)
import re                # for text cleaning using regex
import math              # for probability & perplexity calculation
import numpy as np       # numerical operations
import pandas as pd      # creating tables
from collections import Counter  # counting n-grams

nltk.download('punkt')
```

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

```
corpus = """
Technology is growing rapidly in the modern world. Artificial intelligence and machine learning
are transforming industries such as healthcare, education, finance, and transportation.
Natural language processing helps computers understand human language.
Deep learning models require large datasets for training.

Sports play an important role in maintaining physical and mental health.
Football, cricket, and athletics are popular sports worldwide.
Regular exercise improves concentration and reduces stress.

Politics influences economic growth and public welfare.
Democracy allows citizens to choose their leaders.
Good governance promotes transparency and accountability.

Health awareness is essential for a balanced lifestyle.
Eating nutritious food and maintaining hygiene prevents diseases.
Mental health is as important as physical health.

Technology, sports, politics, and health together shape society.
""" * 30   # repeated to exceed 1500 words
```

```
   # Prevents zero probability for unseen words or n-grams   print(corpus[:500])
```

```

Technology is growing rapidly in the modern world. Artificial intelligence and machine learning
are transforming industries such as healthcare, education, finance, and transportation.
Natural language processing helps computers understand human language.
Deep learning models require large datasets for training.

Sports play an important role in maintaining physical and mental health.
Football, cricket, and athletics are popular sports worldwide.
Regular exercise improves concentration and reduc
```

```
def preprocess_text(text):               # This function cleans and prepares the raw text
    # convert to lowercase
    text = text.lower()

    # remove numbers and punctuation
    text = re.sub(r'[^a-z\s]', '', text)

    # split into sentences
    sentences = text.split('\n')

    processed_sentences = []
    for sentence in sentences:
        words = sentence.split()
        if len(words) > 0:
            # add start and end tokens
            processed_sentences.append(['<s>'] + words + ['</s>'])

    return processed_sentences
```

```
sentences = preprocess_text(corpus)  # Apply preprocessing
print(sentences[0])
```

```
['<s>', 'technology', 'is', 'growing', 'rapidly', 'in', 'the', 'modern', 'world', 'artificial', 'intelligence', 'and', 'machine'
```

```
unigram_counts = Counter()  # Count frequency of each individual word
for sentence in sentences:
```

```
    unigram_counts.update(sentence)

total_unigrams = sum(unigram_counts.values())
```

```
bigram_counts = Counter()    # Count frequency of consecutive word pairs
unigram_context = Counter()

for sentence in sentences:
    for i in range(len(sentence) - 1):
        bigram = (sentence[i], sentence[i+1])
        bigram_counts[bigram] += 1
        unigram_context[sentence[i]] += 1
```

```
trigram_counts = Counter()      # Count frequency of three consecutive words
bigram_context = Counter()

for sentence in sentences:
    for i in range(len(sentence) - 2):
        trigram = (sentence[i], sentence[i+1], sentence[i+2])
        trigram_counts[trigram] += 1
        bigram_context[(sentence[i], sentence[i+1])] += 1
```

```
def unigram_probability(sentence):     # Prevents zero probability for unseen words or n-grams
    prob = 1
    V = len(unigram_counts) # Vocabulary size for Laplace smoothing
    for word in sentence:
        # Calculate probability for each word with Laplace smoothing
        prob *= (unigram_counts[word] + 1) / (total_unigrams + V)
    return prob


def bigram_probability(sentence):
    prob = 1
    V = len(unigram_counts) # Vocabulary size for Laplace smoothing
    for i in range(len(sentence) - 1):
        bigram = (sentence[i], sentence[i+1]) # Get the current bigram
        # Calculate bigram probability with Laplace smoothing
        prob *= (bigram_counts[bigram] + 1) / (unigram_context[sentence[i]] + V)
    return prob


def trigram_probability(sentence):
    prob = 1
    V = len(unigram_counts) # Vocabulary size for Laplace smoothing
    for i in range(len(sentence) - 2):
        trigram = (sentence[i], sentence[i+1], sentence[i+2]) # Get the current trigram
        # Calculate trigram probability with Laplace smoothing
        prob *= (trigram_counts[trigram] + 1) / (bigram_context[(sentence[i], sentence[i+1])] + V)
    return prob
```

```
test_sentences = [
    "<s> technology improves healthcare </s>".split(),
    "<s> sports improve health </s>".split(),
    "<s> democracy supports growth </s>".split(),
    "<s> artificial intelligence grows fast </s>".split(),
    "<s> nutrition improves mental health </s>".split()
]
```

```
for s in test_sentences:
    print("Sentence:", " ".join(s))
    print("Unigram:", unigram_probability(s))
    print("Bigram:", bigram_probability(s))
    print("Trigram:", trigram_probability(s))
    print()
```

```
Sentence: <s> technology improves healthcare </s>
Unigram: 6.021470510408388e-09
Bigram: 5.5374001452432835e-08
Trigram: 8.230452674897121e-07

Sentence: <s> sports improve health </s>
Unigram: 1.411457093593109e-09
Bigram: 9.536633583474542e-07
Trigram: 1.02880658436214e-06
```

```
Sentence: <s> democracy supports growth </s>
Unigram: 9.871263131817029e-11
Bigram: 4.690147664003873e-08
Trigram: 1.02880658436214e-06

Sentence: <s> artificial intelligence grows fast </s>
Unigram: 2.2232574621209523e-14
Bigram: 5.211275182226526e-10
Trigram: 1.1431184270690446e-08

Sentence: <s> nutrition improves mental health </s>
Unigram: 6.605943381805191e-12
Bigram: 1.8765633825546682e-08
Trigram: 2.83493369913123e-07
```

```
def perplexity(sentence, model):
    N = len(sentence) # Get the length of the sentence
    log_prob = 0 # Initialize log probability

    # Calculate log probability based on the specified model (unigram, bigram, or trigram)
    if model == "unigram":
        for word in sentence:
            log_prob += math.log(unigram_probability([word])) # Sum log probabilities of individual words
    elif model == "bigram":
        log_prob = math.log(bigram_probability(sentence)) # Use bigram probability for the entire sentence
    elif model == "trigram":
        log_prob = math.log(trigram_probability(sentence)) # Use trigram probability for the entire sentence

    # Calculate perplexity using the formula: exp(-log_prob / N)
    return math.exp(-log_prob / N)
```

```
for s in test_sentences:
    print("Sentence:", " ".join(s))
    print("Unigram Perplexity:", perplexity(s, "unigram"))
    print("Bigram Perplexity:", perplexity(s, "bigram"))
    print("Trigram Perplexity:", perplexity(s, "trigram"))
    print()
```

```
Sentence: <s> technology improves healthcare </s>
Unigram Perplexity: 44.06152123963561
Bigram Perplexity: 28.270846883945705
Trigram Perplexity: 16.47840814959176

Sentence: <s> sports improve health </s>
Unigram Perplexity: 58.89337781117068
Bigram Perplexity: 16.00003676939737
Trigram Perplexity: 15.759166826422602

Sentence: <s> democracy supports growth </s>
Unigram Perplexity: 100.25948148909517
Bigram Perplexity: 29.225550255449416
Trigram Perplexity: 15.759166826422602

Sentence: <s> artificial intelligence grows fast </s>
Unigram Perplexity: 188.58263812685047
Bigram Perplexity: 35.25136991157913
Trigram Perplexity: 21.069365756459884

Sentence: <s> nutrition improves mental health </s>
Unigram Perplexity: 73.00358576081786
Bigram Perplexity: 19.398707458264983
Trigram Perplexity: 12.337945539935726
```