```
import re  # re      -> used for text cleaning (remove punctuation, numbers)
import math   # math    -> used for logarithmic and perplexity calculations
from collections import Counter, defaultdict  # Counter -> used to count words and n-grams efficiently (defaultdict is not u
```

```
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 to create a sufficiently large dataset
```

```
print(corpus[:500])  # Helps verify that the dataset is loaded correctly by showing the first 500 characters
```

```

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 for n-gram analysis
    # convert all text to lowercase to ensure consistency
    text = text.lower()

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

    # split the cleaned text into individual sentences based on newline characters
    sentences = text.split('\n')

    processed_sentences = []
    for sentence in sentences:
        words = sentence.split() # split each sentence into words
        if len(words) > 0:
            # add start of sentence (<s>) and end of sentence (</s>) tokens
            # to mark sentence boundaries, which is crucial for n-gram models
            processed_sentences.append(['<s>'] + words + ['</s>'])

    return processed_sentences
```

```
sentences = preprocess_text(corpus)  # Apply the preprocessing function to the corpus
print(sentences[0]) # Print the first processed sentence to verify the output format
```

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

```
unigram_counts = Counter() # Initialize a Counter to store the frequency of each individual word
for sentence in sentences:
    unigram_counts.update(sentence) # Update the counts for all words in each sentence

total_unigrams = sum(unigram_counts.values()) # Calculate the total number of words in the corpus
# This value is used as the denominator for unigram probability calculations and for vocabulary size in smoothing
```

```
bigram_counts = Counter()  # Initialize a Counter for bigram frequencies (pairs of consecutive words)
unigram_context = Counter() # Initialize a Counter for the context of bigrams (the first word in a pair)

for sentence in sentences:
    for i in range(len(sentence) - 1): # Iterate through each word in the sentence, up to the second to last word
        bigram = (sentence[i], sentence[i+1]) # Form a bigram from the current word and the next word
        bigram_counts[bigram] += 1 # Increment the count for this specific bigram
        unigram_context[sentence[i]] += 1 # Increment the count for the first word of the bigram (its context)
```

```
trigram_counts = Counter() # Initialize a Counter for trigram frequencies (sequences of three consecutive words)
bigram_context = Counter() # Initialize a Counter for the context of trigrams (the first two words in a sequence)

for sentence in sentences:
    for i in range(len(sentence) - 2): # Iterate through each word in the sentence, up to the third to last word
        trigram = (sentence[i], sentence[i+1], sentence[i+2]) # Form a trigram from the current word and the next two words
        trigram_counts[trigram] += 1 # Increment the count for this specific trigram
        bigram_context[(sentence[i], sentence[i+1])] += 1 # Increment the count for the first two words of the trigram (its
```

```
def unigram_probability(sentence):  # This function calculates the probability of a sentence using a unigram model
# Calculate unigram probability with add-1 smoothing to prevent zero probabilities for unseen words
    prob = 1
    V = len(unigram_counts) # Vocabulary size, used for smoothing
    for word in sentence:
        # (count(word) + 1) / (total_words + V) is the formula for add-1 smoothed unigram probability
        prob *= (unigram_counts[word] + 1) / (total_unigrams + V)
    return prob

def bigram_probability(sentence):
    prob = 1
    V = len(unigram_counts) # Vocabulary size, used for smoothing
    for i in range(len(sentence) - 1):
        bigram = (sentence[i], sentence[i+1]) # Get the current bigram
        # (count(bigram) + 1) / (count(first_word_of_bigram) + V) is the formula for add-1 smoothed bigram probability
        prob *= (bigram_counts[bigram] + 1) / (unigram_context[sentence[i]] + V)
    return prob

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

```
# STEP 9: Define test sentences for evaluation
test_sentences = [
    "<s> technology improves healthcare </s>".split(), # A test sentence related to technology and health
    "<s> sports improve health </s>".split(),        # A test sentence related to sports and health
    "<s> democracy supports growth </s>".split(),     # A test sentence related to politics and economy
    "<s> artificial intelligence grows fast </s>".split(), # A test sentence on AI
    "<s> nutrition improves mental health </s>".split() # A test sentence on health and well-being
]
```

```
# STEP 10: Compute and print the probabilities of each test sentence using unigram, bigram, and trigram models
for s in test_sentences:
    print("Sentence:", " ".join(s)) # Print the current test sentence
    print("Unigram:", unigram_probability(s)) # Calculate and print its unigram probability
    print("Bigram:", bigram_probability(s))   # Calculate and print its bigram probability
    print("Trigram:", trigram_probability(s)) # Calculate and print its trigram probability
    print() # Print a blank line for better separation between sentences
```

```
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
```

```
# Perplexity measures how well the model predicts the sentence (lower perplexity indicates a better model)
def perplexity(sentence, model):
    N = len(sentence) # N is the number of words in the sentence (including start/end tokens)
    log_prob = 0 # Initialize log probability to accumulate values

    if model == "unigram":
        # For unigram, perplexity is typically calculated on individual word probabilities
        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 the pre-calculated bigram probability for the entire senter
    elif model == "trigram":
        log_prob = math.log(trigram_probability(sentence)) # Use the pre-calculated trigram probability for the entire sent

    # Perplexity formula: exp(- (1/N) * log(P(sentence)))
    return math.exp(-log_prob / N)
```

```
# STEP 12: Compute and print the perplexity of each test sentence for unigram, bigram, and trigram models
for s in test_sentences:
    print("Sentence:", " ".join(s)) # Print the current test sentence
    print("Unigram Perplexity:", perplexity(s, "unigram")) # Calculate and print unigram perplexity
    print("Bigram Perplexity:", perplexity(s, "bigram"))   # Calculate and print bigram perplexity
    print("Trigram Perplexity:", perplexity(s, "trigram")) # Calculate and print trigram perplexity
    print() # Print a blank line for better separation
```

```
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
```