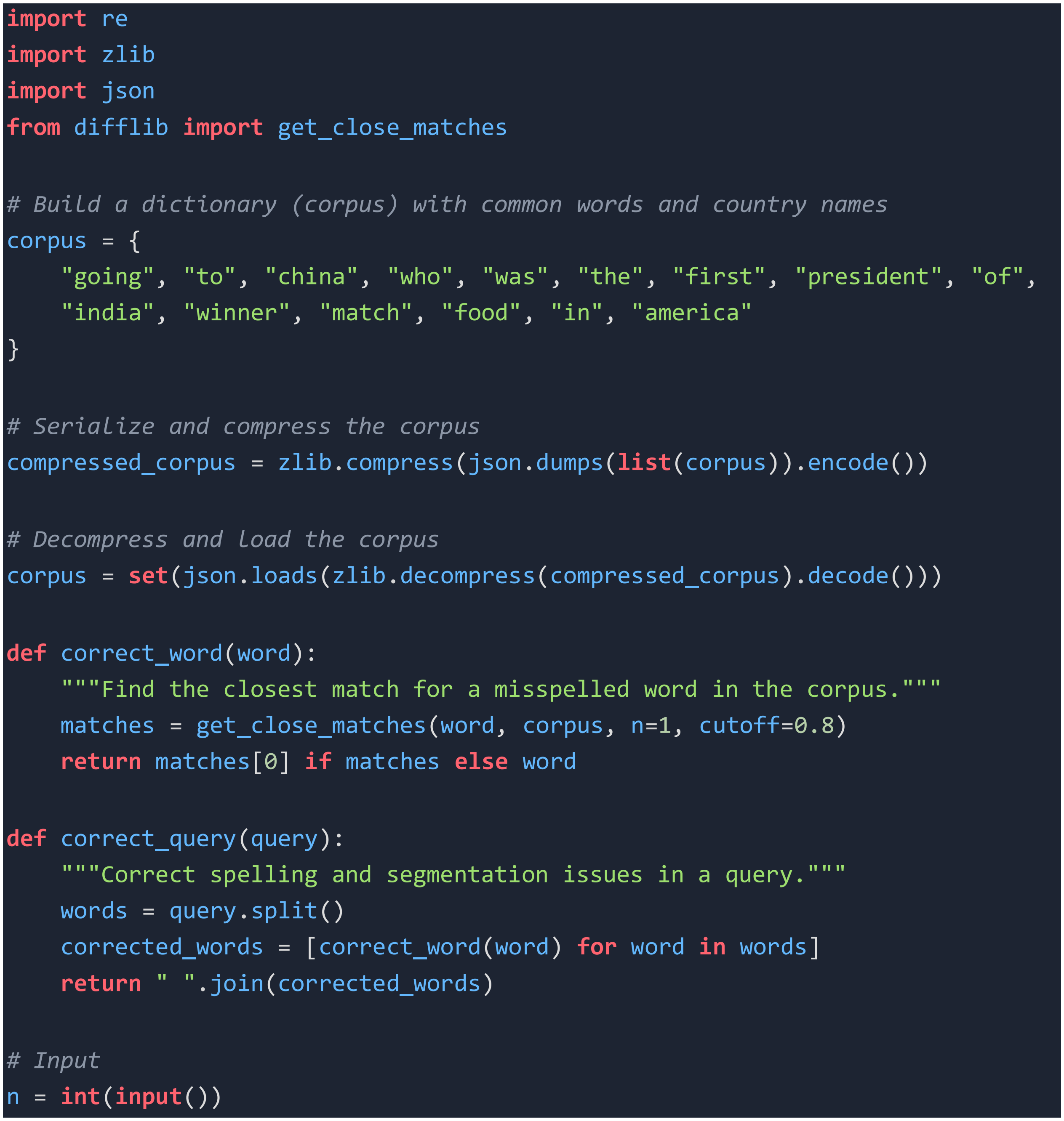
NLP HOLIDAY ASSIGNMENT:

Name : N.Manasa Priya
Roll No :2211CS020366
Section : AIML-Theta

1. Correct the Search Query

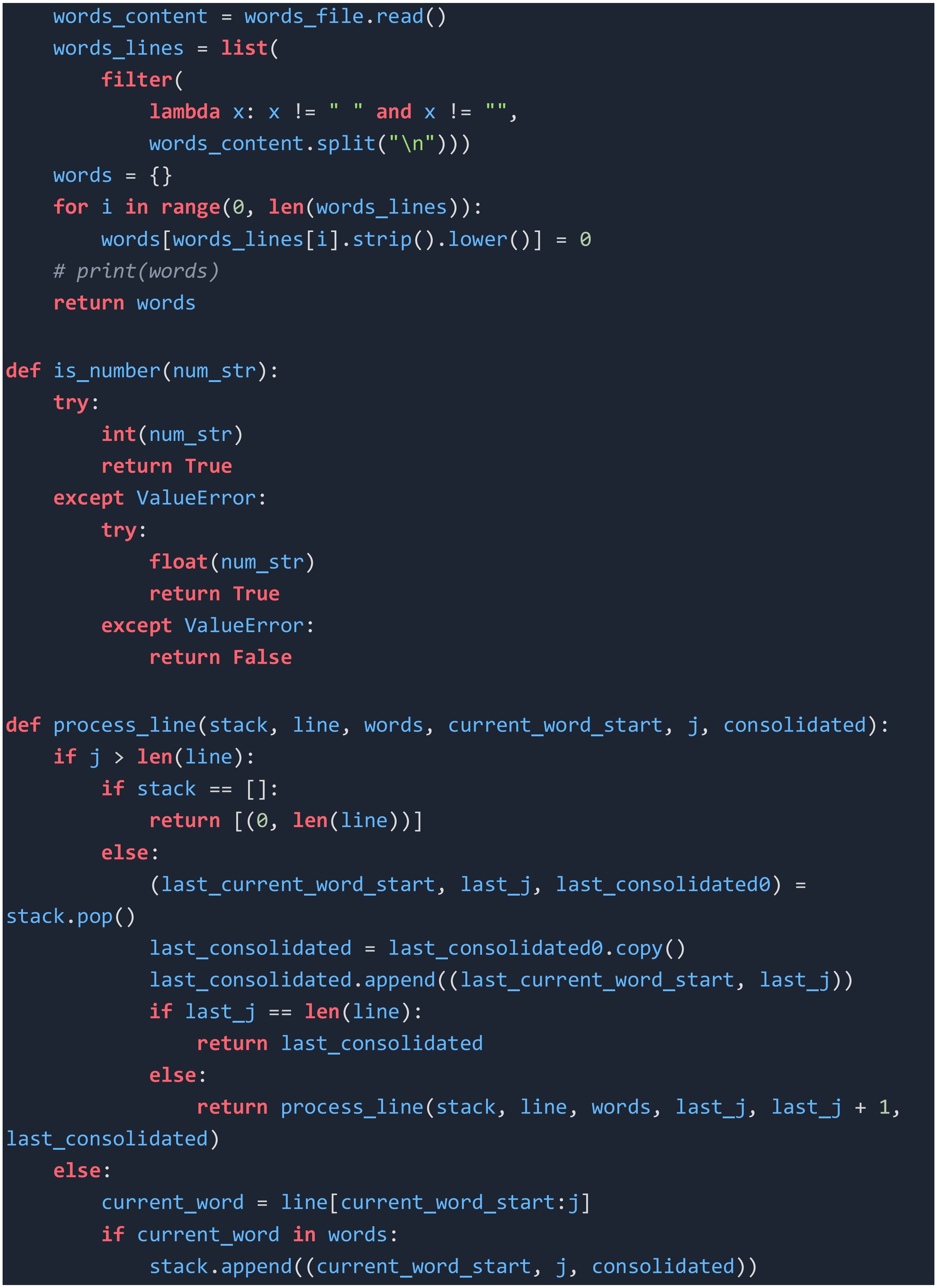


```python
import re
import zlib
import json
from difflib import get_close_matches

# Build a dictionary (corpus) with common words and country names
corpus = {
    "going", "to", "china", "who", "was", "the", "first", "president", "of",
    "india", "winner", "match", "food", "in", "america"
}

# Serialize and compress the corpus
compressed_corpus = zlib.compress(json.dumps(list(corpus)).encode())

# Decompress and load the corpus
corpus = set(json.loads(zlib.decompress(compressed_corpus).decode()))

def correct_word(word):
    """Find the closest match for a misspelled word in the corpus."""
    matches = get_close_matches(word, corpus, n=1, cutoff=0.8)
    return matches[0] if matches else word

def correct_query(query):
    """Correct spelling and segmentation issues in a query."""
    words = query.split()
    corrected_words = [correct_word(word) for word in words]
    return " ".join(corrected_words)

# Input
n = int(input())
```

```
queries = [input().strip() for _ in range(n)]

# Output corrected queries
for query in queries:
    print(correct_query(query))
```

2. Deterministic Url and HashTag Segmentation

```
import sys

def clean_url(line0):
    i = 0
    line = line0[4:] if line0[0:4] == "www." else line0
    rline = list(reversed(line))
    last_dot = 0
    while i < len(line):
        if rline[i] == '.':
            if i > 0 and i < len(line) - 1:
                if rline[i - 1].isalpha():
                    last_dot = i
            else:
                last_dot = i
        i+=1
    return line[0:len(line) - (last_dot+1)]



def clean(lines):
    return [
            line[1:].strip().lower()
        if
            line[0] == '#'
        else
            clean_url(line).strip().lower()
        for line in lines
    ]

def get_words():
    words_file = open("words.txt",'r')
```



```
    words_content = words_file.read()
    words_lines = list(
        filter(
            lambda x: x != " " and x != "",
            words_content.split("\n")))
    words = {}
    for i in range(0, len(words_lines)):
        words[words_lines[i].strip().lower()] = 0
    # print(words)
    return words

def is_number(num_str):
    try:
        int(num_str)
        return True
    except ValueError:
        try:
            float(num_str)
            return True
        except ValueError:
            return False

def process_line(stack, line, words, current_word_start, j, consolidated):
    if j > len(line):
        if stack == []:
            return [(0, len(line))]
        else:
            (last_current_word_start, last_j, last_consolidated0) =
stack.pop()
            last_consolidated = last_consolidated0.copy()
            last_consolidated.append((last_current_word_start, last_j))
            if last_j == len(line):
                return last_consolidated
            else:
                return process_line(stack, line, words, last_j, last_j + 1,
last_consolidated)
    else:
        current_word = line[current_word_start:j]
        if current_word in words:
            stack.append((current_word_start, j, consolidated))
        elif is_number(current_word):
            stack.append((current_word_start, j, consolidated))
        return process_line(stack, line, words, current_word_start, j + 1,
consolidated)




def separate(lines, words):
    for line in lines:
        line_words = process_line([], line, words, 0, 1, [])
        line_words_print = []
        for (start, end) in line_words:
            line_words_print.append(line[start:end])
        print(" ".join(line_words_print))

if __name__ == '__main__':
    words = get_words()
    s = sys.stdin.read()
    lines = list(
        filter(
            lambda x: x != " " and x != "",
            s.split("\n")))
    cleaned_lines = clean(lines[1:])
    # print(cleaned_lines)
    separate(cleaned_lines, words)
    # print(separated)
```

3. Disambiguation: Mouse vs Mouse

```
import json

# Offline model building
animal_words = ["tail", "fur", "genome", "species", "temperature",
"postnatal"]
computer_words = ["device", "click", "cursor", "input", "keyboard", "screen"]

model = {
    "animal": list(animal_words),  # Convert to list for JSON serialization
    "computer-mouse": list(computer_words)  # Convert to list for JSON serialization
}

# Serialize the model (done offline)
with open("mouse_context_model.json", "w") as f:
    json.dump(model, f)

# Function to classify sentences
def classify_sentence(sentence, model):
    words = sentence.lower().split()
    animal_score = sum(1 for word in words if word in model["animal"])
    computer_score = sum(1 for word in words if word in model["computer-mouse"])

    return "animal" if animal_score > computer_score else "computer-mouse"

# Load the model (during runtime)
with open("mouse_context_model.json", "r") as f:
    loaded_model = json.load(f)

# Input processing
n = int(input())  # Number of sentences
results = []
for _ in range(n):
    sentence = input().strip()
    results.append(classify_sentence(sentence, loaded_model))

# Output results
for result in results:
    print(result)
```

4. Language Detection

```
# Define common words for each language
common_words = {
    'English': ['the', 'is', 'in', 'and', 'to', 'a', 'of', 'that', 'it', 'with'],
    'French': ['le', 'est', 'dans', 'et', 'a', 'un', 'de', 'que', 'il', 'avec'],
    'German': ['der', 'ist', 'in', 'und', 'zu', 'ein', 'von', 'dass', 'es', 'mit'],
    'Spanish': ['el', 'es', 'en', 'y', 'a', 'un', 'de', 'que', 'lo', 'con', 'si', 'quieres'," te "
    "tienes" "poner" "las" "pilas."]
}

# Function to detect the language based on common words
def detect_language(text):
    # Remove non-ASCII characters and convert to lowercase
    text = ''.join([char for char in text if ord(char) < 128])

    # Split the text into words, removing punctuation
    words = text.lower().split()

    # Create a dictionary to store the count of common words for each language
    word_count = {'English': 0, 'French': 0, 'German': 0, 'Spanish': 0}

    # Count how many common words appear for each language
    for word in words:
        for language, word_list in common_words.items():
            if word in word_list:
                word_count[language] += 1

    # Find the language with the highest count of common words
    detected_language = max(word_count, key=word_count.get)

    # Output the detected language
    print(detected_language)

# Read the entire input until EOF
import sys
text = sys.stdin.read().strip()  # Read everything from stdin and remove extra spaces/newlines
```

```
# Detect the language
detect_language(text)
```

5. The Missing Apostrophes

```
import re

# Function to insert apostrophes in missing places
def insert_apostrophes(text):
    # Handle common contractions correctly
    contractions = {
        "dont": "don't", "doesnt": "doesn't", "cant": "can't", "wont":
"won't", "isnt": "isn't",
        "arent": "aren't", "im": "I'm", "ill": "I'll", "theres": "there's",
"its": "it's",
        "whats": "what's", "whos": "who's", "thats": "that's", "youre":
"you're", "were": "we're",
        "theyre": "they're", "hasnt": "hasn't", "havent": "haven't", "hadnt":
"hadn't", "couldnt": "couldn't",
        "wouldnt": "wouldn't", "shouldnt": "shouldn't", "wasnt": "wasn't",
"didnt": "didn't",
        "hed": "he'd", "id": "I'd", "wed": "we'd", "theyve": "they've",
"youve": "you've",
        "ive": "I've", "youd": "you'd", "shes": "she's", "hes": "he's",
"were": "we're"
    }

    # Replace contractions
    for key, value in contractions.items():
        text = re.sub(r'\b' + key + r'\b', value, text)

    # Handle possessive cases: "party's" instead of "partys"
    # Don't touch plural forms: "parties" remains "parties"
    text = re.sub(r'(\w+?)s\b', r'\1\'s', text)

    return text
```

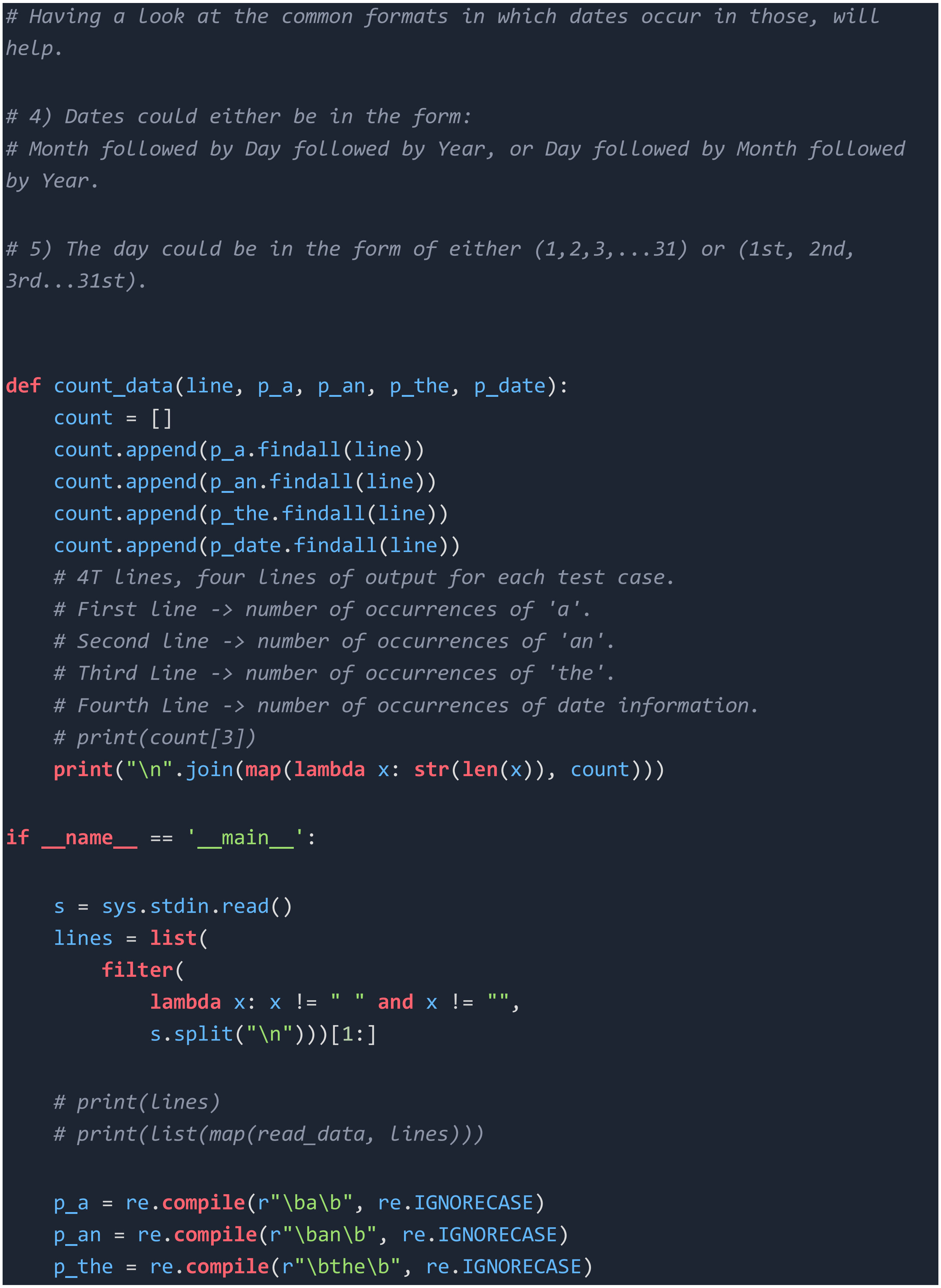
```python
# Sample input
input_text = """At a news conference Thursday at the Russian manned-space
facility in Baikonur, Kazakhstan, Kornienko said "we will be missing nature,
we will be missing landscapes, woods." He admitted that on his previous trip
into space in 2010 "I even asked our psychological support folks to send me a
calendar with photographs of nature, of rivers, of woods, of lakes."
Kelly was asked if hed miss his twin brother Mark, who also was an astronaut.
"Were used to this kind of thing," he said. "Ive gone longer without seeing
him and it was great."
The mission wont be the longest time that a human has spent in space - four
Russians spent a year or more aboard the Soviet-built Mir space station in
the 1990s."""

# Insert apostrophes
output_text = insert_apostrophes(input_text)

# Print the output with apostrophes inserted
print(output_text)
```

6.Segment the Twitter Hashtags

```python
# A sample list of common words (In practice, you would use a much larger
dictionary)
common_words = set([
    "we", "are", "the", "people", "mention", "your", "faves", "now",
"playing",
    "dead", "follow", "me", "walking", "fave", "is", "a", "to", "and", "this"
])

# Function to segment a single hashtag
def segment_hashtag(hashtag):
    n = len(hashtag)
    dp = [None] * (n + 1)
    dp[0] = []  # Base case: an empty string has a valid segmentation

    # Iterate through all possible end points of the substring
    for i in range(1, n + 1):
        for j in range(i):
            word = hashtag[j:i]
            if word in common_words and dp[j] is not None:
                dp[i] = dp[j] + [word]
                break  # If we found a valid split, no need to check further for this 'i'

    # If dp[n] is not None, we have a valid segmentation
    return " ".join(dp[n]) if dp[n] else hashtag

# Read input
n = int(input())  # Read the number of hashtags
for _ in range(n):
    hashtag = input().strip()
    print(segment_hashtag(hashtag))
```

7. A Text-Processing Warmup

```
import sys
import re



# '15/11/2012','15/11/12', '15th March 1999','15th March 99' or '20th of March, 1999').

# You can make the following assumptions

# 1) In the date, year and day will always be in numeric form.
# Which means, you don't have to worry about "fifteenth" or "twentieth" etc.
# Month, could be either numeric form (1-12) or with its name (January-December, Jan-Dec).

# 2) This is a bit open ended, and somewhat intentionally so.
# The aim is for you to try to write something which figures out as many common patterns as possible, in which dates are present in text.

# 3) Most of the test cases are Wikipedia articles.
```



```
# Having a look at the common formats in which dates occur in those, will
help.

# 4) Dates could either be in the form:
# Month followed by Day followed by Year, or Day followed by Month followed
by Year.

# 5) The day could be in the form of either (1,2,3,...31) or (1st, 2nd,
3rd...31st).


def count_data(line, p_a, p_an, p_the, p_date):
    count = []
    count.append(p_a.findall(line))
    count.append(p_an.findall(line))
    count.append(p_the.findall(line))
    count.append(p_date.findall(line))
    # 4T lines, four lines of output for each test case.
    # First line -> number of occurrences of 'a'.
    # Second line -> number of occurrences of 'an'.
    # Third Line -> number of occurrences of 'the'.
    # Fourth Line -> number of occurrences of date information.
    # print(count[3])
    print("\n".join(map(lambda x: str(len(x)), count)))

if __name__ == '__main__':

    s = sys.stdin.read()
    lines = list(
        filter(
            lambda x: x != " " and x != "",
            s.split("\n")))[1:]

    # print(lines)
    # print(list(map(read_data, lines)))

    p_a = re.compile(r"\ba\b", re.IGNORECASE)
    p_an = re.compile(r"\ban\b", re.IGNORECASE)
    p_the = re.compile(r"\bthe\b", re.IGNORECASE)
    months =
"January|February|March|April|May|June|July|August|September|October|November
|December"
    months3 = "Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec"
    allmonths = months + "|" + months3
    p1str = r"\d\d?/\d\d?/\d\d?\d?\d?"
    p2str = r"(\d\d?)(st|nd|rd|th)?\s+(of\s+)?(" + allmonths +
r")(\s*,)?\s+(\d\d?\d?\d?)"
    p3str = "(" + allmonths +
r")\s+(\d\d?)(st|nd|rd|th)?(\s*,)?\s+(\d\d?\d?\d?)"
    pstr = "(" + "|".join([p1str, p2str, p3str]) + ")"
    p_date = re.compile(pstr)

    for l in lines:
        count_data(l, p_a, p_an, p_the, p_date)
```

8.Who is it?

```
import re

# Read input values
n = int(input())  # Number of lines of text
text = [input().strip() for _ in range(n)]  # Text lines
noun_phrases = input().strip().split(';')  # List of noun phrases

# Combine all text into a single string
full_text = ' '.join(text)

# Find all pronouns surrounded by '**'
pronouns_with_positions = list(re.finditer(r'\*\*(\w+)\*\*', full_text))

# List to store resolved pronouns
resolved_pronouns = []
```

```
noun_phrase_index = 0  # To track the noun phrases used

# Loop over the pronouns found in the text and resolve them
for match in pronouns_with_positions:
    pronoun = match.group(1)  # Extract the pronoun (e.g., "she", "her",
"it")

    if noun_phrase_index < len(noun_phrases):  # Ensure there are still noun
phrases available
        resolved_pronouns.append(noun_phrases[noun_phrase_index])
        noun_phrase_index += 1  # Move to the next noun phrase

# Output the resolved pronouns
for resolved in resolved_pronouns:
    print(resolved)
```

9.Expand the Acronyms

```
import re

def extract_expansions(snippets):
    expansions = {}

    # Regex for acronyms followed by (expansion) and acronym followed by "is
expansion"
    pattern1 = re.compile(r'([A-Z]{2,})\s*\(([^)]+)\)')
    pattern2 = re.compile(r'([A-Z]{2,})\s+is\s+([^\.]+)')

    for snippet in snippets:
        # Matches for acronym in parentheses
        matches1 = pattern1.findall(snippet)
        for acronym, expansion in matches1:
            expansions[acronym] = expansion.strip()

        # Matches for acronym followed by "is"
```

```python
        matches2 = pattern2.findall(snippet)
        for acronym, expansion in matches2:
            expansions[acronym] = expansion.strip()

    return expansions

def main():
    # Reading input
    import sys
    input = sys.stdin.read
    data = input().split("\n")

    N = int(data[0])  # Number of snippets
    snippets = data[1:N+1]  # N snippets
    tests = data[N+1:]  # Test acronyms

    expansions = extract_expansions(snippets)

    # Output the expansion for each test acronym
    for test in tests:
        if test:
            print(expansions.get(test, "Expansion not found"))

if __name__ == "__main__":
    main()
```