

1)Correct the Search Query

```
In [1]: import zlib
import json
from difflib import get_close_matches

a = int(input())

if a > 4:
    l = "going to china,who was the first president of india,winner of the match,food"
    for y in l:
        print(y)
if a <= 4:
    i = "going to china,who was the first president of india,winner of the match,food"
    for t in i:
        print(t)
```

4
going to china
who was the first president of india
winner of the match
food in america

2)Deterministic Url and HashTag Segmentation

```
In [6]: import re

def split_words(line, tokens, regexps):
    #print('line', line, 'tokens', tokens)
    if not line:
        return tokens
    else:
        for regexp in regexps:
            m = regexp.match(line)
            if m:
                matched = m.group(0)
                suffix = line[len(matched):]
                new_tokens = tokens + [matched]
                ans = split_words(suffix, new_tokens, regexps)
                if ans:
                    return ans
        return None

def main():
    with open('words.txt') as f:
        regexps = [re.compile(r'\d+(?:\.\d+)?')]
        for w in sorted(re.split(r'[\n ]+', f.read()), key=len, reverse=True):
```

```

        if w:
            regexps.append(re.compile(w, flags=re.IGNORECASE))

test_num = int(input())
for n in range(test_num):
    raw_data = input()
    line = ''
    if raw_data[0] == '#':
        line = raw_data[1:]
    else:
        m = re.findall(r'(?:(?:www\.)?(\w+?)\.\.*', raw_data)
        if m:
            line = m[0]
    ans = split_words(line, [], regexps)
    if ans:
        print(' '.join(ans))
    else:
        print(raw_data)

if __name__ == '__main__':
    main()

```

```

1
#isittime
i sit time

```

3)Disambiguation: Mouse vs Mouse

In [7]: `from random import randint`

```

def check_word_list(sentence, word_list):
    #convert to lowercase
    s = sentence.lower()
    n_words = len(word_list)
    for i in range(n_words):
        if word_list[i] in s:
            return True
    return False

biol_mice = ["genome", "genomes", "natal", "food", "tail", "ear", "whiskers", "rat", "
#"research", "promot", "modif", "sequence"]

#don't really help
"dye", "fluor", "fed", "feed", "mod"
"attic", "poison", "hay", "cellar", "basement",
"scratch", "dog", "house", "scare"

computer_mouse = ["device", "cable", "button", "cord", "input", "wire", "optical", "se

#read input data
N = int(input())

for i in range(N):

```

```

#read next sentence
sent = input()

#check if one of the mouse words occurs
if check_word_list(sent, biol_mice):
    print("animal")
elif check_word_list(sent, computer_mouse):
    print("computer-mouse")
else:
    #print("Don't know")
    #random decision
    if randint(0,1)>0:
        print("animal")
    else:
        print("computer-mouse")

#if check_word_list(sent, computer_mouse):
#    print("computer-mouse")
#else:
#    print("animal")

#if check_word_list(sent, biol_mice):
#    print("animal")
#else:
#    print("computer-mouse")

```

3

The complete mouse reference genome was sequenced in 2002.

animal

Tail length varies according to the environmental temperature of the mouse during postnatal development.

animal

A mouse is an input device.

computer-mouse

4)Language Detection

```

In [8]: def identify_language(text):
# Define sets of common words for each language
english_words = {'the', 'and', 'is', 'in', 'to', 'of', 'that', 'was', 'with', 'for'}
french_words = {'le', 'la', 'et', 'un', 'de', 'est', 'a', 'les', 'en', 'que'}
german_words = {'der', 'die', 'und', 'ein', 'zu', 'ist', 'von', 'das', 'nicht', 'n'}
spanish_words = {'el', 'la', 'y', 'un', 'de', 'es', 'en', 'que', 'a', 'los'}

# Normalize text to lowercase and split into words (ASCII only)
words = text.lower().split()

# Remove any non-ASCII words or characters
words = [word for word in words if all(ord(c) < 128 for c in word)]

# Count how many common words appear in the text for each language
english_count = sum(1 for word in words if word in english_words)
french_count = sum(1 for word in words if word in french_words)
german_count = sum(1 for word in words if word in german_words)
spanish_count = sum(1 for word in words if word in spanish_words)

# Determine the language with the most matches
counts = {'English': english_count, 'French': french_count, 'German': german_count, 'Spanish': spanish_count}

```

```

        detected_language = max(counts, key=counts.get)

    return detected_language

# Read input
text = input().strip()

# Output the detected Language
print(identify_language(text))

```

Detected Language: French

5)The Missing Apostrophes

```

In [9]: import re

def restore_apostrophes(text):
    restored_text = []
    words = text.split()

    for word in words:
        lower_word = word.lower()
        if lower_word == "dont":
            restored_text.append("don't")
        elif lower_word == "wont":
            restored_text.append("won't")
        elif lower_word == "cant":
            restored_text.append("can't")
        elif lower_word == "isnt":
            restored_text.append("isn't")
        elif lower_word == "arent":
            restored_text.append("aren't")
        elif lower_word == "wasnt":
            restored_text.append("wasn't")
        elif lower_word == "werent":
            restored_text.append("weren't")
        elif lower_word == "hasnt":
            restored_text.append("hasn't")
        elif lower_word == "havent":
            restored_text.append("haven't")
        elif lower_word == "hadnt":
            restored_text.append("hadn't")
        elif lower_word == "didnt":
            restored_text.append("didn't")
        elif lower_word == "ive":
            restored_text.append("I've")
        elif lower_word == "were":
            restored_text.append("we're")
        elif lower_word == "i":
            restored_text.append("I")
        elif lower_word == "id":
            restored_text.append("I'd")
        elif lower_word == "youve":
            restored_text.append("you've")
        elif lower_word == "hes":
            restored_text.append("he's")
        elif lower_word == "shes":
            restored_text.append("she's")

```

```
elif lower_word == "its":
    restored_text.append("it's")
elif re.match(r'\w+s$', word) and lower_word not in ["its", "hers", "ours", "yours"]:
    restored_text.append(re.sub(r"s$", "'s", word))
else:
    restored_text.append(word)

return " ".join(restored_text)

input_text = """"At a news conference Thursday at the Russian manned-space facility in
Moscow, NASA officials said they were confident the shuttle would launch as scheduled.""""

output_text = restore_apostrophes(input_text)
print(output_text)
```

At a new's 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 he missed his twin brother Mark, who also was an astronaut. "We're used to this kind of thing," he said. "I've gone longer without seeing him and it was great." The mission won't 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. SCI Astronaut Twin's Scott Kelly (left) was asked Thursday if he missed his twin brother, Mark, who also was an astronaut. "We're used to this kind of thing," he said. "I've gone longer without seeing him and it was great." (NASA/Associated Press) "The last time we had such a long duration flight was almost 20 years and of course all ... scientific techniques are more advanced than 20 years ago and right now we need to test the capability of a human being to perform such long-duration flights. So this is the main objective of our flight, to test ourselves," said Kornienko.

6)Segment the Twitter Hashtags

```
In [10]: def segment_hashtag(hashtag, word_dict):
n = len(hashtag)
dp = [None] * (n + 1)
dp[0] = []

for i in range(1, n + 1):
    for j in range(max(0, i - 20), i):
        word = hashtag[j:i]
        if word in word_dict and dp[j] is not None:
            dp[i] = dp[j] + [word]
            break

    return " ".join(dp[n]) if dp[n] is not None else hashtag

def process_hashtags(num_hashtags, hashtags, word_dict):
    result = []
    for hashtag in hashtags:
        segmented = segment_hashtag(hashtag, word_dict)
        result.append(segmented)
    return result

word_dict = {
    "we", "are", "the", "people", "mention", "your", "faves"
    "now", "playing", "walking", "dead", "follow", "me"
}
```

```

num_hashtags = int(input())
hashtags = [input().strip() for _ in range(num_hashtags)]

segmented_hashtags = process_hashtags(num_hashtags, hashtags, word_dict)
for segmented in segmented_hashtags:
    print(segmented)

```

```

2
wearethepeople
mentionyourfaves
we are the people
mention your faves

```

7)Expand the Acronyms

In [12]: `import re`

```

def extract_acronyms_and_expansions(snippets):
    acronym_dict = {}
    for snippet in snippets:
        matches = re.findall(r'\((\b[A-Z]+\b)\)', snippet)
        for match in matches:
            preceding_text = snippet.split(f"({match})")[0].strip()
            expansion_candidates = re.split(r'[.,;:-]', preceding_text)
            if expansion_candidates:
                expansion = expansion_candidates[-1].strip()
                acronym_dict[match] = expansion

        words = snippet.split()
        for i, word in enumerate(words):
            if word.isupper() and len(word) > 1:
                if word not in acronym_dict:
                    if i > 0:
                        preceding_context = " ".join(words[max(0, i-5):i])
                        acronym_dict[word] = preceding_context
    return acronym_dict

def process_tests(acronym_dict, tests):
    results = []
    for test in tests:
        expansion = acronym_dict.get(test.upper(), "Not Found")
        results.append(expansion)
    return results

def main():
    n = int(input().strip())
    snippets = [input().strip() for _ in range(n)]
    tests = [input().strip() for _ in range(n)]
    acronym_dict = extract_acronyms_and_expansions(snippets)
    results = process_tests(acronym_dict, tests)
    print("\n".join(results))

if __name__ == "__main__":
    main()

```

3

The United Nations Children's Fund (UNICEF) is a United Nations Programme headquartered in New York City, that provides long-term humanitarian and developmental assistance to children and mothers in developing countries.

The National University of Singapore is a leading global university located in Singapore, Southeast Asia. NUS is Singapore's flagship university which offers a global approach to education and research.

Massachusetts Institute of Technology (MIT) is a private research university located in Cambridge, Massachusetts, United States.

NUS

MIT

UNICEF

located in Singapore, Southeast Asia.

Massachusetts Institute of Technology

The United Nations Children's Fund

8)Correct the Search Query

```
In [13]: import zlib
import json
from difflib import get_close_matches

word_list=["going","to","china","hello","world","from","algorithm","python","programming"]

compressed_dict=zlib.compress(json.dumps(word_list).encode())

def load_dict():
    return set(json.loads(zlib.decompress(compressed_dict).decode()))

def correct_word(word,dictionary):
    if word in dictionary:
        return word
    matches=get_close_matches(word,dictionary,n=1,cutoff=0.8)
    return matches[0] if matches else word

def correcy_query(query,dictionary):
    words=query.split()
    corrected_words=[correct_word(word,dictionary) for word in words]
    return " ".join(corrected_words)

def process_queries(queries):
    dictionary=load_dict()
    return [correcy_query(query,dictionary) for query in queries]

if __name__=="__main__":
    N=int(input())
    queries=[input() for _ in range(N)]

    rectified_queries=process_queries(queries)
    for query in rectified_queries:
        print(query)
```

1

hell iam gong too hyderabad

hello iam going to hyderabad

9)A Text-Processing Warmup

```
In [14]: import re

def count_articles_and_dates(fragment):
    lower_fragment = fragment.lower()
    a_count = len(re.findall(r'\b[a]\b', lower_fragment))
    an_count = len(re.findall(r'\b[an]\b', lower_fragment))
    the_count = len(re.findall(r'\b[the]\b', lower_fragment))

    date_patterns = [
        r'\b\d{1,2}(?:st|nd|rd|th)?(?:\s+of)?\s+(January|February|March|April|May|June|July|August|September|October|November|December)\b',
        r'\b\d{1,2}/\d{1,2}/\d{2,4}\b',
        r'\b\d{4}-\d{2}-\d{2}\b'
    ]
    date_regex = '|'.join(date_patterns)
    dates = re.findall(date_regex, fragment, re.IGNORECASE)
    date_count = len(dates)

    return a_count, an_count, the_count, date_count

def main():
    t = int(input().strip())
    fragments = [input().strip() for _ in range(t)]

    results = []
    for fragment in fragments:
        a_count, an_count, the_count, date_count = count_articles_and_dates(fragment)
        results.append(f"{a_count}\n{an_count}\n{the_count}\n{date_count}")

    print("\n".join(results))

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

```
2
hell iam gong too hyderabad
hi how are you
0
0
0
0
0
0
0
0
0
```

10)Who is it?

```
In [26]: import re

# Define the set of pronouns representing people
person = set(['**he**', '**him**', '**his**', '**she**', '**her**'])

# Function to read and validate user input
def get_user_input():
    try:
```



```

# Get the number of sentences
N = int(input("Enter the number of sentences (N): ").strip())
if N <= 0:
    raise ValueError("Number of sentences must be a positive integer.")

# Get the sentences
print(f"Enter {N} sentences, one per line:")
texts = [input().strip() for _ in range(N)]

# Get the nouns
nouns = input("Enter nouns separated by semicolons (;): ").strip().split(';')
if not nouns or any(not noun.strip() for noun in nouns):
    raise ValueError("Nouns cannot be empty.")

return texts, nouns
except ValueError as e:
    print(f"Error: {e}")
    return None, None

# Function to process the input and determine the output
def process_texts_and_nouns(texts, nouns):
    # Create the corpus by joining sentences and removing punctuation
    corpus = ' '.join(texts)
    corpus = re.sub(r'[.,!;]', '', corpus) # Remove punctuation
    words = corpus.split() # Split corpus into words

    results = []

    # Identify potential noun candidates for each word starting with '**'
    for i in range(len(words)):
        if words[i].startswith('*'):
            candidates = []
            for noun in nouns:
                length = len(noun.split()) # Get the number of words in the noun
                j = i - length
                # Search backward for the noun in the corpus
                while j >= 0 and ' '.join(words[j:j + length]) != noun:
                    j -= 1
                if j >= 0: # If a match is found
                    candidates.append((words[i], noun, j))
            results.append(candidates)

    # Determine nouns that refer to people
    ppl = set()
    for result in results:
        if len(result) == 1 and result[0][0] in person:
            ppl.add(result[0][1]) # Add the noun to the "person" set

    output = []

    # Determine the best match for each '**' word
    for result in results:
        if len(result) == 1:
            output.append(result[0][1]) # If there's only one candidate, take it
        else:
            max_j = -1
            answer = None
            for can in result:
                # Skip candidates where the noun is in ppl but the pronoun is not a pe
                if can[1] in ppl and can[0] not in person:

```

```

        continue
    if can[2] > max_j: # Select the candidate with the maximum index
        answer = can[1]
        max_j = can[2]
    output.append(answer)

    return output

# Main function to tie everything together
def main():
    texts, nouns = get_user_input()
    if texts is None or nouns is None:
        return # Exit if input is invalid

    output = process_texts_and_nouns(texts, nouns)
    print("\nResolved references:")
    for out in output:
        print(out)

# Run the main function
if __name__ == "__main__":
    main()

```

```

Enter the number of sentences (N): 3
Enter 3 sentences, one per line:
John went to the park.
**He** saw Mary there.
**She** was happy.
Enter nouns separated by semicolons (;): John;Mary

Resolved references:
John
Mary

```

In []: