```python
# coding: utf-8

####################################################
################### Question 1 ###################
####################################################

input_string="cat sat on the mat"
input_list = input_string.split() # put string into a list
print input_list

N=2 # the gram number, you can change it to 1, 2, 3, 4 to get different output

#xrange(int) is to iterate numbers from 0 to int
for i in xrange(len(input_list)-N+1):
    print input_list[i:i+N] # print the answer

####################################################
################### Question 2 ###################
####################################################

output = "The gunman was shot dead by police ."  # translation output
reference = "The gunman was shot dead by the police ." # reference

# this function is reused from Question 1, to shown n-gram string
def gram_scanner(n, input_string): # n is the number of gram; input_string is a string type
    result_list=[]
    input_list = input_string.split()
    for i in xrange(len(input_list)-n+1):
        result_list.append(input_list[i:i+n])
    return result_list # return them in list type

# to count how many matched tokens (include punctuations) in n-gram
def calculate_match(output_gram, reference_gram): # output_gram and reference_gram are the output and reference in n-gram format (list type)
    matched_number = 0 # initialization
    for o in output_gram:
        matched = [i for i,x in enumerate(reference_gram) if x == o] # enumerate() is to get the index of current element of the list

        if matched != []:
            matched_number+=1
            del reference_gram[matched[0]] # remove the matched token from reference
    return matched_number

# to calculate precision; N is the gram number; p1 is precision in 1-gram

N=1
output_gram = gram_scanner(N, output)
reference_gram = gram_scanner(N, reference)
p1 = calculate_match(output_gram,reference_gram)/float(len(output_gram))

N=2
output_gram = gram_scanner(N, output)
reference_gram = gram_scanner(N, reference)
p2 = calculate_match(output_gram,reference_gram)/float(len(output_gram))

N=3
output_gram = gram_scanner(N, output)
reference_gram = gram_scanner(N, reference)
p3 = calculate_match(output_gram,reference_gram)/float(len(output_gram))

N=4
output_gram = gram_scanner(N, output)
reference_gram = gram_scanner(N, reference)
p4 = calculate_match(output_gram,reference_gram)/float(len(output_gram))

p = p1*p2*p3*p4

# to calculate Brevity Penalty
BP = min(1,len(output.split())/float(len(reference.split())))
print BP
```

```

# final result
import math
print math.pow(p, 1.0 / 4) * BP # match.pow() is the way to do evolution calculation (same as java)

##################################################
################### Question 3 ###################
###################  Method 1  ###################
##################################################

output = "The gunman was shot dead by police ."  # translation output
reference_1 = "The gunman was shot dead by the police ." # reference 1
reference_2 = "The gunman was shot dead by the police ." # reference 2
reference_3 = "Police killed the gunman ." # reference 3
reference_4 = "The gunman was shot dead by the police ." # reference 4
reference = [reference_1,reference_2,reference_3,reference_4] # put all references into a list

# same as Question 2
def gram_scanner(n, input_string):
    result_list=[]
    input_list = input_string.split()
    for i in xrange(len(input_list)-n+1):
        result_list.append(input_list[i:i+n])
    return result_list

# same as Question 2
def calculate_match(output_gram,reference_gram):
    matched_number = 0
    for o in output_gram:
        matched = [i for i,x in enumerate(reference_gram) if x == o]
        if matched != []:
            matched_number+=1
            del reference_gram[matched[0]]
    return matched_number

# to calculate precision with multi-reference; N is the gram number; p1 is precision in 1-gram
N=1
output_gram = gram_scanner(N, output)

# use a loop go through each reference in the reference list, and calculate match and precision
correct_list = []
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    correct_list.append(calculate_match(output_gram,reference_gram))
p1 = max(correct_list)/float(len(output_gram))
print p1

N=2
output_gram = gram_scanner(N, output)
correct_list = []
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    correct_list.append(calculate_match(output_gram,reference_gram))
p2 = max(correct_list)/float(len(output_gram))

N=3
output_gram = gram_scanner(N, output)
correct_list = []
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    correct_list.append(calculate_match(output_gram,reference_gram))
p3 = max(correct_list)/float(len(output_gram))

N=4
output_gram = gram_scanner(N, output)
correct_list = []
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    correct_list.append(calculate_match(output_gram,reference_gram))
p4 = max(correct_list)/float(len(output_gram))

print p1,p2,p3,p4
p = p1*p2*p3*p4

# to calculate Brevity Penalty
ref_len_list = [len(r.split()) for r in reference]
ref_len = min(ref_len_list, key=lambda x:abs(x-len(output.split())))# select the length of one reference, whose length is most close to the output length

# same as Question 2
BP = min(1,len(output.split())/float(ref_len))
print BP

# same as Question 2
import math
print math.pow(p, 1.0 / 4) * BP

##################################################
################### Question 3 ###################
###################  Method 2  ###################
##################################################

output = "the the the gunman was shot dead by police ."
reference_1 = "the gunman was shot dead by the police ."
reference_2 = "the gunman was shot dead by the police ."
reference_3 = "police killed the gunman ."
reference_4 = "the gunman was shot dead by the police ."

reference = [reference_1,reference_2,reference_3,reference_4]

def count_references_keys(refs_gram_list):
    combined_references_keys = list(set([item for sublist in refs_gram_list for item in sublist]))
    references_key_count_dic={}
    for key in combined_references_keys:
        counts=[]
        for each_reference in refs_gram_list:
            counts.append(each_reference.count(key))
        references_key_count_dic[key]=max(counts)
    return references_key_count_dic

def gram_scanner(n, input_string):
    result_list=[]
    input_list = input_string.split()
    for i in xrange(len(input_list)-n+1):
        result_list.append(' '.join(input_list[i:i+n]))
    return result_list

def calculate_match(output_gram,references_gram_dic):
    print output_gram,references_gram_dic
    matched_number = 0
    for token in output_gram:
        if references_gram_dic.has_key(token) and references_gram_dic[token]>0:
            matched_number+=1
            references_gram_dic[token] -= 1
    return matched_number

N=1
output_gram = gram_scanner(N, output)
references_gram_list = []
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    references_gram_list.append(reference_gram)
references_gram_dic = count_references_keys(references_gram_list)
match_number = calculate_match(output_gram, references_gram_dic)
p1 = match_number/float(len(output_gram))

N=2
output_gram = gram_scanner(N, output)
references_gram_list = []
```

```
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    references_gram_list.append(reference_gram)
references_gram_dic = count_references_keys(references_gram_list)
match_number = calculate_match(output_gram, references_gram_dic)
p2 = match_number/float(len(output_gram))

N=3
output_gram = gram_scanner(N, output)
references_gram_list = []
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    references_gram_list.append(reference_gram)
references_gram_dic = count_references_keys(references_gram_list)
match_number = calculate_match(output_gram, references_gram_dic)
p3 = match_number/float(len(output_gram))

N=4
output_gram = gram_scanner(N, output)
references_gram_list = []
for ref in reference:
    reference_gram = gram_scanner(N, ref)
    references_gram_list.append(reference_gram)
references_gram_dic = count_references_keys(references_gram_list)
match_number = calculate_match(output_gram, references_gram_dic)
p4 = match_number/float(len(output_gram))

print p1,p2,p3,p4
p = p1*p2*p3*p4

ref_len_list = [len(r.split()) for r in reference]
ref_len = min(ref_len_list, key=lambda x:abs(x-len(output.split())))

BP = min(1,len(output.split())/float(ref_len))
print BP

import math
print math.pow(p, 1.0 / 4) * BP


```