Get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself

Sample String : 'restart' Expected Result : 'resta$t'

In [1]:
```
def change_char(str1):
  char = str1[0]
  str1 = str1.replace(char, '$')
  str1 = char + str1[1:]

  return str1

print(change_char('restart'))
```

```
resta$t
```

Minimum Gifts

A Company has decided to give some gifts to all of its employees. For that, the company has given some rank to each employee. Based on that rank, the company has made certain rules for distributing the gifts.

The rules for distributing the gifts are:

Each employee must receive at least one gift.

Employees having higher ranking get a greater number of gifts than their neighbours.

What is the minimum number of gifts required by the company?

Constraints

$1 < T < 10$

$1 < N < 100000$

$1 < Rank < 10^9$

Input

The first line contains integer T, denoting the number of test cases.

For each test case:

The first line contains integer N, denoting the number of employees.

The second line contains N space-separated integers, denoting the rank of each employee.

Output

For each test case print the number of minimum gifts required on a new line.

Example 1

Input

2

5

1 2 1 5 2

2

1 2

Output

7

3

Explanation

For test case 1, adhering to the rules mentioned above,

Employee # 1 whose rank is 1 gets one gift

Employee # 2 whose rank is 2 gets two gifts

Employee # 3 whose rank is 1 gets one gift

Employee # 4 whose rank is 5 gets two gifts

Employee # 5 whose rank is 2 gets one gift

Therefore, total gifts required is 1 + 2 + 1 + 2 + 1 = 7

Similarly, for test case 2, adhering to the rules mentioned above,

Employee # 1 whose rank is 1 gets one gift

Employee # 2 whose rank is 2 gets two gifts

Therefore, the total gifts required is 1 + 2 =3

Possible solution:

Input:

2

5

1 2 1 5 2

2

1 2

In [2]:
```
t = int(input())
while t>0:
    n = int(input())
    a = list(map(int, input().split()))
    gifts = [1]
    #gifts.append(1)
    for i in range(1, n):
      if a[i] > a[i-1]:
        gifts.append(gifts[i-1]+1)
      else:
        gifts.append(1)
    #print(gifts)
    for i in range(n-2,-1,-1):
      if a[i]>a[i+1]:
        gifts[i] = max (1 + gifts[i+1], gifts[i])
    g = 0
    print(gifts)
    for i in range(n):
      g += gifts[i]
    print(g)
    t-=1
```

```
1
3
5 3 7
[2, 1, 2]
5
```

Write a Python program to create a Caesar encryption.

Note: In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence.

key would be in negative if you want to left shift. key would be positive if you want to right shift.

1st Method

```
In [59]: s=input()
         key=int(input())
         s1=""
         for i in s:
             if i>='A' and i<='Z':
                 if ord(i)+key>ord('Z'):
                     i=chr(ord(i)+key-26)
                 else:
                     i=chr(ord(i)+key)
                 s1+= i
             elif i>='a' and i<='z':
                 if ord(i)+key>ord('z'):
                     i=chr(ord(i)+key-26)
                 else:
                     i=chr(ord(i)+key)
                 s1+= i
             else:
                 s1+= i
         print(s1)
```

```
thisispython
2
vjkukuravjqp
```

2nd Method

In [58]:
```
def caesar_encrypt(realText, step):
    outText = []
    cryptText = []

    uppercase = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U'
    lowercase = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u'

    for eachLetter in realText:
        if eachLetter in uppercase:
            index = uppercase.index(eachLetter)
            crypting = (index + step) % 26
            cryptText.append(crypting)
            newLetter = uppercase[crypting]
            outText.append(newLetter)
        elif eachLetter in lowercase:
            index = lowercase.index(eachLetter)
            crypting = (index + step) % 26
            cryptText.append(crypting)
            newLetter = lowercase[crypting]
            outText.append(newLetter)
    return ''.join(outText)
code = caesar_encrypt('thisispython', 2)
print()
print(code)
print()
```

```
vjkukuravjqp
```

3rd Method

In [61]:
```python
def encrypt(key, message):
    message = message.upper()
    alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    result = ""

    for letter in message:
        if letter in alpha: #if the letter is actually a letter
            #find the corresponding ciphertext letter in the alphabet
            letter_index = (alpha.find(letter) + key) % len(alpha)

            result = result + alpha[letter_index]
        else:
            result = result + letter

    return result

def decrypt(key, message):
    message = message.upper()
    alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    result = ""

    for letter in message:
        if letter in alpha: #if the letter is actually a letter
            #find the corresponding ciphertext letter in the alphabet
            letter_index = (alpha.find(letter) - key) % len(alpha)

            result = result + alpha[letter_index]
        else:
            result = result + letter

    return result
message="ywa"
print(encrypt(4, message))
print(decrypt(4, 'cae'))
```

```
CAE
YWA
```

Demo of Join Function

In [4]:
```
#demo of join function
#if joins all arguments given inside of list of strings to a blank string.
a="hello"
b="LJU"
print(''.join([a,b]))
```

```
helloLJU
```

In [5]:
```
#demo of join function
#if joins all arguments given inside of list of strings with space in between.
a="hello"
b="LJU"
c='welcome'
print(' '.join([a,b,c]))
```

```
hello LJU welcome
```

Write a Python program to reverse words in a string

In [9]:
```
def reverse_string_words(text):
    for line in text.split('\n'):
        return(' '.join(line.split()[::-1]))
print(reverse_string_words("The quick brown fox jumps over the lazy dog."))
print(reverse_string_words("Python Exercises."))
```

```
dog. lazy the over jumps fox brown quick The
Exercises. Python
```

Write a Python program to find the second most repeated word in a given string.

In [15]:
```python
def word_count(str):
    counts = dict()
    words = str.split()

    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1

    counts_x = sorted(counts.items(), key=lambda kv: kv[1])
    print(counts_x)
    return counts_x[-2]

print(word_count("""Both of these issues are fixed by postponing the evaluation of annotations.
                 Instead of compiling code which executes expressions in annotations at their definition time,
                 the compiler stores the annotation in a string form equivalent to the AST of the expression in question
                 If needed, annotations can be resolved at runtime using typing.get_type_hints().
                 In the common case where this is not required, the annotations are cheaper to store
                 (since short strings are interned by the interpreter) and make startup time faster."""))
```

```
[('Both', 1), ('these', 1), ('issues', 1), ('fixed', 1), ('postponing', 1), ('evaluation', 1), ('annotations.', 1), ('I
nstead', 1), ('compiling', 1), ('code', 1), ('which', 1), ('executes', 1), ('expressions', 1), ('their', 1), ('definiti
on', 1), ('time,', 1), ('compiler', 1), ('stores', 1), ('annotation', 1), ('a', 1), ('string', 1), ('form', 1), ('equiv
alent', 1), ('AST', 1), ('expression', 1), ('question.', 1), ('If', 1), ('needed,', 1), ('can', 1), ('be', 1), ('resolv
ed', 1), ('runtime', 1), ('using', 1), ('typing.get_type_hints().', 1), ('In', 1), ('common', 1), ('case', 1), ('wher
e', 1), ('this', 1), ('is', 1), ('not', 1), ('required,', 1), ('cheaper', 1), ('store', 1), ('(since', 1), ('short',
1), ('strings', 1), ('interned', 1), ('interpreter)', 1), ('and', 1), ('make', 1), ('startup', 1), ('time', 1), ('faste
r.', 1), ('by', 2), ('at', 2), ('to', 2), ('are', 3), ('in', 3), ('annotations', 3), ('of', 4), ('the', 8)]
('of', 4)
```

Write a Python program to create a string from two given strings concatenating uncommon characters of the said strings

```
In [16]: def uncommon_chars_concat(s1, s2):

             set1 = set(s1)
             set2 = set(s2)

             common_chars = list(set1 & set2)
             result = [ch for ch in s1 if ch not in common_chars] + [ch for ch in s2 if ch not in common_chars]
             return(''.join(result))

         s1 = 'abcdpqr'
         s2 = 'xyzabcd'
         print("Original Substrings:\n",s1+"\n",s2)
         print("\nAfter concatenating uncommon characters:")
         print(uncommon_chars_concat(s1, s2))
```

```
Original Substrings:
 abcdpqr
 xyzabcd

After concatenating uncommon characters:
pqrxyz
```

Write a Python program to find the minimum window in a given string which will contain all the characters of another given string. (You can use your own method to find the solution as well)

Input : str1 = " PRWSOERIUSFK "

str2 = " OSU "

Output: Minimum window is "OERIUS

In [25]:
```
def minWindow(s,t) :

        counter_t = {}
        counter_s = {}

        for c in t:
            counter_t[c] = counter_t.get(c, 0) + 1


        i = 0
        j = 0

        left = -1
        right = -1

        valid = 0

        for i in range(len(s)):

            while j < len(s) and valid < len(counter_t):
                counter_s[s[j]] = counter_s.get(s[j], 0) + 1

                if s[j] in counter_t and counter_s[s[j]] == counter_t[s[j]]:
                    valid += 1


                j += 1


            if valid == len(counter_t):
                if left == -1 or j - i < right - left:
                    left = i
                    right = j


            counter_s[s[i]] -= 1
            if s[i] in counter_t and counter_s[s[i]] == counter_t[s[i]] - 1:
                valid -= 1

        if left == -1:
            return "not found"
        return s[left : right]
s="1a2b3cbva"
t="abc"
print(minWindow(s,t))
s = "PRWSvOERIUSFK"
t = "OSU"
print(minWindow(s,t))
s="abcfghioi9i99i9i9ijnbhhjkmijnbhfi  hklij %^^hilkiujj hjmjkfrtygh000000000ijnhjhijhhgfghjmnbvghoiuhghgfij hjkjij"
t="hijkwxy"
print(minWindow(s,t))
```

```
cbva
OERIUS
not found
```

Write a Python program to remove unwanted characters from a given string.

```
Sample Output:

    Original String : Pyth*^on Exercis^es

        After removing unwanted characters:

            Python Exercises

            Original String : A%^!B#*CD

After removing unwanted characters:

    ABCD
```

In [32]:
```
def remove_chars(str1, unwanted_chars):
    for i in unwanted_chars:
        str1 = str1.replace(i, '')
    return str1



str1 = "Pyth*^on Exercis^es"
str2 = "A%^!B#*CD"

unwanted_chars = ["#", "*", "!", "^", "%"]
print ("Original String : " + str1)
print("After removing unwanted characters:")
print(remove_chars(str1, unwanted_chars))
print ("\nOriginal String : " + str2)
print("After removing unwanted characters:")
print(remove_chars(str2, unwanted_chars))
```

```
Original String : Pyth*^on Exercis^es
After removing unwanted characters:
Python Exercises

Original String : A%^!B#*CD
After removing unwanted characters:
ABCD
```

Write a Python program to remove punctuations from a given string.

Sample Output:

Original text:

String! With. Punctuation?

After removing Punctuations from the said string:

String With Punctuation

In [33]:
```
def remove_punctuations(text):
    punc_list = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
    result = ""
    for char in text:
        if char not in punc_list:
            result = result + char
    return result
text = "@^&$String! With.-- Punctuation?"
print("Original text:")
print(text)
result = remove_punctuations(text)
print("\nAfter removing Punctuations from the said string:")
print(result)
```

```
Original text:
@^&$String! With.-- Punctuation?

After removing Punctuations from the said string:
String With Punctuation
```

Write a Python program to remove repeated consecutive characters and replace with the single letters and print new updated string.

Sample Data:

("Red Green White") -> "Red Gren White"

("aabbbcdeffff") -> "abcdef"

("Yellowwooddoor") -> "Yelowodor"

In [34]:
```
def test(text):
  result = []
  for x in text:
    if not result or result[-1] != x:
      result.append(x)
  return ''.join(result)

text ="Red Green White"
print("Original string:", text)
print("Remove repeated consecutive characters and replace with the single letters:")
print(test(text))
text = "aabbbcdeffff"
print("\nOriginal string:", text)
print("Remove repeated consecutive characters and replace with the single letters:")
print(test(text))
text = "Yellowwooddoor"
print("\nOriginal string:", text)
print("Remove repeated consecutive characters and replace with the single letters:")
print(test(text))
```

```
Original string: Red Green White
Remove repeated consecutive characters and replace with the single letters:
Red Gren White

Original string: aabbbcdeffff
Remove repeated consecutive characters and replace with the single letters:
abcdef

Original string: Yellowwooddoor
Remove repeated consecutive characters and replace with the single letters:
Yelowodor
```

In [35]:
```
def test(text):
    return ''.join(text[i] for i in range(len(text)) if i==0 or text[i-1]!=text[i])
text ="Red Green White"
print("Original string:", text)
print("Remove repeated consecutive characters and replace with the single letters:")
print(test(text))
text = "aabbbcdeffff"
print("\nOriginal string:", text)
print("Remove repeated consecutive characters and replace with the single letters:")
print(test(text))
text = "Yellowwooddoor"
print("\nOriginal string:", text)
print("Remove repeated consecutive characters and replace with the single letters:")
print(test(text))
```

```
Original string: Red Green White
Remove repeated consecutive characters and replace with the single letters:
Red Gren White

Original string: aabbbcdeffff
Remove repeated consecutive characters and replace with the single letters:
abcdef

Original string: Yellowwooddoor
Remove repeated consecutive characters and replace with the single letters:
Yelowodor
```

Write a Python program to that takes two strings. Count the number of times each string contains the same three letters at the same index. Go to the editor

Sample Data:

("Red RedGreen") -> 1

("Red White Red White) -> 7

("Red White White Red") > 0

In [36]:
```
def test(text1, text2):
           ctr = 0
           for i in range(len(text1) - 2):
                       if text1[i:i+3] == text2[i:i+3]:
                                   ctr += 1
           return ctr
text1 ="Red"
text2 ="RedGreen"
print("Original strings:", text1,text2)
print("Check said two strings contain three letters at the same index:")
print(test(text1, text2))
text1 ="Red White"
text2 ="Red White"
print("Original strings:", text1,text2)
print("Check said two strings contain three letters at the same index:")
print(test(text1, text2))
text1 ="Red White"
text2 ="White Red"
print("Original strings:", text1,text2)
print("Check said two strings contain three letters at the same index:")
print(test(text1, text2))
```

```
Original strings: Red RedGreen
Check said two strings contain three letters at the same index:
1
Original strings: Red White Red White
Check said two strings contain three letters at the same index:
7
Original strings: Red White White Red
Check said two strings contain three letters at the same index:
0
```

Write a Python script to sort (ascending and descending) a dictionary by value.

In [37]:
```
def sort_dict_by_value(d, reverse = False):
  return dict(sorted(d.items(), key = lambda x: x[1], reverse = reverse))
print("Original dictionary elements:")
colors = {'Red': 1, 'Green': 3, 'Black': 5, 'White': 2, 'Pink': 4}
print(colors)
print("\nSort (ascending) the said dictionary elements by value:")
print(sort_dict_by_value(colors))
print("\nSort (descending) the said dictionary elements by value:")
print(sort_dict_by_value(colors, True))
```

```
Original dictionary elements:
{'Red': 1, 'Green': 3, 'Black': 5, 'White': 2, 'Pink': 4}

Sort (ascending) the said dictionary elements by value:
{'Red': 1, 'White': 2, 'Green': 3, 'Pink': 4, 'Black': 5}

Sort (descending) the said dictionary elements by value:
{'Black': 5, 'Pink': 4, 'Green': 3, 'White': 2, 'Red': 1}
```

I have a string of words. I would like to sort the list by the length of each word so that the longest word is at the top

In [1]:
```
s="nbhg mjknh kiuytgft nbh nbhgtygnm mnjku lkmnhg nbh nb ghyuytredf bghyyt kmnjkmnjkmn bghbghb vfgbvc hgv bvg"
d={}
for i in s.split():
    d[i]=len(i)
#print(d)
l1=d.values()
l2=d.keys()
#print(l1)
#print(l2)
ans=[i[1] for i in sorted(list(zip(l1,l2)),reverse=True)]
print(ans)
```

```
['kmnjkmnjkmn', 'ghyuytredf', 'nbhgtygnm', 'kiuytgft', 'bghbghb', 'vfgbvc', 'lkmnhg', 'bghyyt', 'mnjku', 'mjknh', 'nbh
g', 'nbh', 'hgv', 'bvg', 'nb']
```

This is a Python Program to create a dictionary with key as first character and value as words starting with that character.

```
In [ ]: s="acd bcd avf ght dfg bgh kju avf ghv bvf ujh kju bgh avf nhj bgh bvf cdf avg sde kju gty frt der cdf xsd zse"
        d={}
        for i in s.split():
            if i[0] not in d.keys():
                d[i[0]]=[]
                d[i[0]].append(i)
            else:
                d[i[0]].append(i)
        print(d)
```

This is a Python Program to create a dictionary with key as character and value as character count repeated in string.first five more repeated character

```
In [39]: s="acd bcd avf ght dfg bgh kju avf ghv bvf ujh kju bgh avf nhj bgh bvf cdf avg sde kju gty frt der cdf xsd zse"
         d={}
         for i in s.split():
             d[i[0]]=d.get(i[0],0)+1
         print(d)
         y=sorted(d.items(),key=lambda x:x[1],reverse=True)
         for i in range(5):
             print(y[i])
```

```
{'a': 5, 'b': 6, 'g': 3, 'd': 2, 'k': 3, 'u': 1, 'n': 1, 'c': 2, 's': 1, 'f': 1, 'x': 1, 'z': 1}
('b', 6)
('a', 5)
('g', 3)
('k', 3)
('d', 2)
```

Sort Dictionary key and values List.

Example 1:

Input:

{'c': [3], 'b': [12, 10], 'a': [19, 4]}

Output:

{'a': [4, 19], 'b': [10, 12], 'c': [3]}

Example 2:

Input:

{'c': [10, 34, 3]}

Output:

{'c': [3, 10, 34]}

In [40]:
```
d = {'c': [3], 'b': [12, 10], 'a': [19, 4]}

res = {}

for i in sorted(d):
  res[i] = sorted(d[i])

print(res)
```

```
{'a': [4, 19], 'b': [10, 12], 'c': [3]}
```

Convert a list of Tuples into Dictionary.

```
Example 1:
```

Input:

[("akash", 10), ("gaurav", 12), ("anand", 14), ("suraj", 20), ("akhil", 25), ("ashish", 30)]

Output:

{'akash': [10], 'gaurav': [12], 'anand': [14], 'suraj': [20], 'akhil': [25], 'ashish': [30]}

Example 2:

Input:

[('A', 1), ('B', 2), ('C', 3)]

Output:

{'A': [1], 'B': [2], 'C': [3]}

In [41]:
```python
L1 = [("akash", 10), ("gaurav", 12), ("anand", 14), ("suraj", 20), ("akhil", 25), ("ashish", 30)]

L = [('A', 1), ('B', 2), ('C', 3)]

d = {}

for i,j in L:
  d[i] = [j]

print(d)
```

```
{'A': [1], 'B': [2], 'C': [3]}
```

Key with maximum unique values

Given a dictionary with values list, extract key whose value has most unique values.

Example 1:

Input:

test_dict = {"CampusX" : [5, 7, 9, 4, 0], "is" : [6, 7, 4, 3, 3], "Best" : [9, 9, 6, 5, 5]}

Output:

CampusX

Example 2:

Input:

test_dict = {"CampusX" : [5, 7, 7, 7, 7], "is" : [6, 7, 7, 7], "Best" : [9, 9, 6, 5, 5]}

Output:

Best

In [42]:
```
test_dict = {"CampusX" : [5, 7, 7, 7, 7], "is" : [6, 7, 7, 7], "Best" : [9, 9, 6, 5, 5]}

max_val = 0
max_key = ''
for i in test_dict:
  if max_val < len(set(test_dict[i])):
    max_val = len(set(test_dict[i]))
    max_key = i

print(max_key)
```

```
Best
```

find uncommon words from two Strings. Statement: Given two sentences as strings A and B. The task is to return a list of all uncommon words. A word is uncommon if it appears exactly once in any one of the sentences, and does not appear in the other sentence. Note: A sentence is a string of space-separated words. Each word consists only of lowercase letters.

Example 1:

Input:

A = "apple banana mango" B = "banana fruits mango"

In [3]:
```
# Code here
A = "apple banana mango"
B = "banana fruits mango"

L = []

for i in A.split():
  if i not in B and i not in L:
    L.append(i)

for j in B.split():
  if j not in A and j not in L:
    L.append(j)

print(L)
```

```
['apple', 'fruits']
```

Check whether the string is Symmetrical. Statement: Given a string. the task is to check if the string is symmetrical or not. A string is said to be symmetrical if both the halves of the string are the same.

Example 1:

Input

khokho Output

The entered string is symmetrical

```
In [5]: s = input('enter the string')

        if len(s)%2 == 0:
          s1 = s[0:len(s)//2]
          s2 = s[len(s)//2:]
        else:
          s1 = s[0:len(s)//2]
          s2 = s[len(s)//2 + 1:]
        if s1 == s2:
          print('symmetrical')
        else:
          print('not symmetrical')
```

```
enter the stringKHOKHO
symmetrical
```

Take a alphanumeric string input and print the sum and average of the digits that appear in the string, ignoring all other characters. Input:

hel123O4every093

Output:

Sum: 22 Avg: 3.14

In [6]:
```
s = 'hel12304every093'

sum = 0
count = 0

for i in s:
  if i.isdigit():
    sum = sum + int(i)
    count += 1

print(sum)
print(sum/count)
print(count)
```

```
22
3.142857142857143
7
```

Create Short Form from initial character

Given a string create short form ofthe string from Initial character. Short form should be capitalised.

Example:

Input:

Data science mentorship program Output:

DSMP

In [7]:
```
inp = 'Data science Mentorship Program'

res = ''

for i in inp.split():
  res = res + i[0].upper()

print(res)
```

```
DSMP
```

A simple way of encrypting a message is to rearrange its characters. One way to rearrange the characters is to pick out the characters at even indices, put them first in the encrypted string, and follow them by the odd characters. For example, the string message would be encrypted as msaeesg because the even characters are m, s, a, e (at indices 0, 2, 4, and 6) and the odd characters are e, s, g (at indices 1, 3, and 5).

(1). Write a program that asks the user for a string and uses this method to encrypt the string.

(2). Write a program that decrypts a string that was encrypted with this method.

In [6]:
```
def encrypt():#to put all our code in function
    strings=input('what is your message: ')
    even=strings[::2]#extract the even part
    odd=strings[1::2]#extract the odd part
    print(even+odd)#print the result
encrypt()

def decrypt():
    string=input('what is your message: ')
    length=len(string)#to get the length of the input
    half_length=(length+1)//2 #half of the input
    even=string[:half_length]#even part
    odd=string[half_length:]#odd part
#here we start inserting each of the part to form original
    msg=''
    for i in range (half_length):
        join=even[i:i+1]+odd[i:i+1]
        msg=msg+join
    print(msg)
decrypt()
```

```
what is your message: This is python programming
Ti spto rgamnhsi yhnpormig
what is your message: Ti spto rgamnhsi yhnpormig
This is python programming
```

A more general version of the above technique is the rail fence cipher, where instead of breaking things into evens and odds, they are broken up by threes, fours or something larger. For instance, in the case of threes, the string secret message would be broken into three groups. The first group is sr sg, the characters at indices 0, 3, 6, 9 and 12. The second group is eemse, the characters at indices 1, 4, 7, 10, and 13. The last group is ctea, the characters at indices 2, 5, 8, and 11. The encrypted message is sr sgeemsectea.

Write a program that asks the user for a string, and an integer determining whether to break things up by threes, fours, or whatever user inputs. Encrypt the string using above method.

Write a decryption program for the same general case. Taking input of any encrypted string from user with number used to break things apart during encryption.

Encryption

In [7]:
```
s=input()
s1=""
key=int(input())
for i in range(key):
    s1+=s[i::key]
print(s1)
```

```
This is python, a programming language
5
Tit omlahshagiagi o rnnespnpagg y,rm u
```

1st Solution for Decryption

In [8]:
```
string=input('what is your message: ')
key=int(input())
length=len(string)#to get the length of the input
part=length//key #half of the input
extra=length%key #extra characters after dividing string in equal parts
part1=string[:(part+1)*extra]
part2=string[(part+1)*extra:]
msg=''
for i in range(part+1):
    if i<part:
        msg+=part1[i::part+1]+part2[i::part]
    else:
        msg+=part1[i::part+1]
print(msg)
```

```
what is your message: Tit omlahshagiagi o rnnespnpagg y,rm u
5
This is python, a programming language
```

2nd Solution for Decryption

In [9]:
```
string=input('what is your message: ')
key=int(input())
length=len(string)#to get the length of the input
part=length//key #half of the input
extra=length%key #extra characters after dividing string in equal parts
part1=string[:(part+1)*extra]
part2=string[(part+1)*extra:]
msg=''
for i in range(part+1):
    if i<part:
        msg+=part1[i::part+1]+part2[i::part]
    else:
        msg+=part1[i::part+1]
print(msg)
```

```
what is your message: Tit omlahshagiagi o rnnespnpagg y,rm u
5
This is python, a programming language
```

Write a Python program which will return the sum of the numbers in the array, returning 0 for an empty array. Except the number 13 is very unlucky, so it does not count and number that come immediately after 13 also do not count.

Example : [1, 2, 3, 4] = 10 [1, 2, 3, 4, 13] = 10 [13, 1, 2, 3, 13] = 5

In [5]:
```
def sum_list(l):
    if len(l)==0:
        return 0
    else:
        sum=0
        for i in range(len(l)):
            if l[i]==13 or l[i-1]==13 and i!=0:
                continue
            else:
                sum+=l[i]
        return sum
l=eval(input("enter list"))
print(sum_list(l))
```

```
enter list[1,13,1,2,13,3,3,13]
6
```

Write Python Program to Add Two Matrices

matrix_1 = [[1, 2, 3],

```
[4, 5, 6],

[7, 8, 9]]
```

matrix_2 = [[1, 2, 3],

```
[4, 5, 6],

[7, 8, 9]]
```

In [7]:
```
matrix_1 = [[1, 2, 3],[4, 5, 6], [7, 8, 9]]
matrix_2 = [[1, 2, 3],[4, 5, 6], [7, 8, 9]]
matrix_result = [[0, 0, 0],[0, 0, 0],[0, 0, 0]]
for rows in range(len(matrix_1)):
    for columns in range(len(matrix_2[0])):
        matrix_result[rows][columns] = matrix_1[rows][columns] + matrix_2[rows][columns]
print("Addition of two matrices is")
print(matrix_result)
for items in matrix_result:
    print(items)
```

```
Addition of two matrices is
[[2, 4, 6], [8, 10, 12], [14, 16, 18]]
[2, 4, 6]
[8, 10, 12]
[14, 16, 18]
```

Program to multiply two matrices using nested for loops

In [8]:
```
#Program to multiply two matrices using nested for loops
# 3x3 matrix
A = [[1,2,3],
     [4,5,6],
     [7,8,9]]

B = [[1,2,3,4],
     [5,6,7,8],
     [2,4,6,8]]

result = [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0]]


for i in range(len(A)):

    for j in range(len(B[0])):

        for k in range(len(B)):
            result[i][j] += A[i][k] * B[k][j]
print(result)
print('Multiplied Matrix:')
for r in result:
    print(r)
```

```
[[17, 26, 35, 44], [41, 62, 83, 104], [65, 98, 131, 164]]
Multiplied Matrix:
[17, 26, 35, 44]
[41, 62, 83, 104]
[65, 98, 131, 164]
```

In [9]:
```
#make the dictionary as key is charcter and value is the count of character in string.after print
#first five max repeated character
s="avbgkbhjdyfbckbvdfgbybncvhbvhcvbmnbvcxasdfghjklpoiuytrewqhjujnbvawdpscmsbndysnvnbvbnfddnbjv"
d={}
for i in s:
    d[i]=d.get(i,0)+1
#print(d)
k=sorted(d.items(),key=lambda x:x[1],reverse=True )
#print(k)
print("first five max repeated character")
for i in range(0,5):
    print(k[i])
```

```
first five max repeated character
('b', 14)
('v', 10)
('n', 8)
('d', 7)
('h', 5)
```

Q. Repeatedly ask the user to enter a team name and how many games the team has won and how many they lost.Store this information in a dictionary where the keys are the team names and the values are a list of the form [wins, losses].

(i) using the dictionary created above, allow the user to enter a team name and print out the team’s winning percentage.

(ii) using dictionary create a list whose entries are the number of wins for each team.

(iii) using the dictionary, create a list of all those teams that have winning records.

In [10]:
```
dic ={}
lstwin = []
lstrec = []
while True :
    name = input ("Enter the name of team (enter q for quit)= ")
    if name == "Q" or name == "q" :
        print()
        break
    else :
        win = int (input("Enter the no.of win match = "))
        loss = int(input("Enter the no.of loss match = "))
        print()
        dic [ name ] = [ win , loss ]
        lstwin += [ win ]
        if win > 0 :
            lstrec += [ name ]

nam = input ("Enter the name of team For Winning = ")
print ("Winning percentage = ",dic [ nam ][0] *100 / (dic [nam ][0] + dic[nam ][1] ))
print()
print("Winning list of all team = ",lstwin)
print("Team who has winning records are ",lstrec)
```

```
Enter the name of team (enter q for quit)= nhm
Enter the no.of win match = 8
Enter the no.of loss match = 2

Enter the name of team (enter q for quit)= nhj
Enter the no.of win match = 7
Enter the no.of loss match = 5

Enter the name of team (enter q for quit)= q

Enter the name of team For Winning = nhm
Winning percentage =  80.0

Winning list of all team =  [8, 7]
Team who has winning records are  ['nhm', 'nhj']
```

Use a list comprehension to produce a list that consists of all palindromic numbers between 100 and 1000.

In [12]:
```
L=[i for i in range(100,1001) if str(i)==str(i)[::-1]]
print(L)
```

```
[101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 202, 212, 222, 232, 242, 252, 262, 272, 282, 292, 303, 313, 323, 33
3, 343, 353, 363, 373, 383, 393, 404, 414, 424, 434, 444, 454, 464, 474, 484, 494, 505, 515, 525, 535, 545, 555, 565, 5
75, 585, 595, 606, 616, 626, 636, 646, 656, 666, 676, 686, 696, 707, 717, 727, 737, 747, 757, 767, 777, 787, 797, 808,
818, 828, 838, 848, 858, 868, 878, 888, 898, 909, 919, 929, 939, 949, 959, 969, 979, 989, 999]
```

Write a program that converts Roman numerals into ordinary numbers. Here are the conversions: M=1000, D=500, C=100, L=50, X=10, V=5 I=1. Don't forget about things like IV being 4 and XL being 40.

Write a program that converts ordinary numbers into Roman numerals

```
In [2]: def romanToInt(s):
                """
                :type s: str
                :rtype: int
                """
                roman = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000,'IV':4,'IX':9,'XL':40,'XC':90,'CD':400,'CM':900}
                i = 0
                num = 0
                while i < len(s):
                    if i+1<len(s) and s[i:i+2] in roman:
                        num+=roman[s[i:i+2]]
                        i+=2
                    else:
                        num+=roman[s[i]]
                        i+=1
                return num
        s=input("Enter number in Roman: ")
        print(romanToInt(s))
```

```
Enter number in Roman: LXIV
64
```

In [14]:
```
def int_to_Roman(num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        roman_num = ''
        i = 0
        while  num > 0:
            for j in range(num // val[i]):
                roman_num += syb[i]
                num -= val[i]
            i += 1
        return roman_num
num=int(input("Input an integer: "))
print(int_to_Roman(num))
```

```
Input an integer: 58
LVIII
```

Create a 5 × 5 list of numbers. Then write a program that creates a dictionary whose keys are the numbers and whose values are the how many times the number occurs. Then print the three most common numbers.

In [15]:
```
L=[[5, 3, 3, 5, 5],
   [3, 2, 4, 3, 3],
   [3, 3, 3, 3, 4],
   [3, 4, 5, 3, 3],
   [5, 4, 1, 2, 3]]
d={}
for i in range(len(L)):
    for j in L[i]:
        d[j]=d.get(j,0)+1
print(d)
L1=[]
for i,j in d.items():
    L1.append([j,i])
L1.sort()
L1.reverse()
L2=[]
for i in range(3):
    print(L1[i][1],'comes',L1[i][0],'times in a 5x5 list')
    L2.append(L1[i][1])
print('Three most common numbers in 5x5 list are', L2)
```

```
{5: 5, 3: 13, 2: 2, 4: 4, 1: 1}
3 comes 13 times in a 5x5 list
5 comes 5 times in a 5x5 list
4 comes 4 times in a 5x5 list
Three most common numbers in 5x5 list are [3, 5, 4]
```

Given an integer value, return a string with the equivalent English text of each digit. For example, an input of 89 results in “eighty-nine” being returned. Restrict values to be between 0 and 1,000.

In [2]:
```
"""Given an int32 number, print it in English."""
def int_to_en(num):
    d = { 0 : 'zero', 1 : 'one', 2 : 'two', 3 : 'three', 4 : 'four', 5 : 'five',
          6 : 'six', 7 : 'seven', 8 : 'eight', 9 : 'nine', 10 : 'ten',
          11 : 'eleven', 12 : 'twelve', 13 : 'thirteen', 14 : 'fourteen',
          15 : 'fifteen', 16 : 'sixteen', 17 : 'seventeen', 18 : 'eighteen',
          19 : 'nineteen', 20 : 'twenty',
          30 : 'thirty', 40 : 'forty', 50 : 'fifty', 60 : 'sixty',
          70 : 'seventy', 80 : 'eighty', 90 : 'ninety' }
    k = 1000

    assert(0 <= num)

    if (num < 20):
        return d[num]

    if (num < 100):
        if num % 10 == 0: return d[num]
        else: return d[num // 10 * 10] + '-' + d[num % 10]

    if (num < k):
        if num % 100 == 0: return d[num // 100] + ' hundred'
        else: return d[num // 100] + ' hundred and ' + int_to_en(num % 100)

print(int_to_en(200))
```

```
two hundred
```

Given an integer value, return a string with the equivalent English text of each digit. For example, an input of 89 results in “eighty-nine” being returned. Restrict values to be between 0 and 10^15.

In [16]:
```
def int_to_en(num):
    d = { 0 : 'zero', 1 : 'one', 2 : 'two', 3 : 'three', 4 : 'four', 5 : 'five',
          6 : 'six', 7 : 'seven', 8 : 'eight', 9 : 'nine', 10 : 'ten',
          11 : 'eleven', 12 : 'twelve', 13 : 'thirteen', 14 : 'fourteen',
          15 : 'fifteen', 16 : 'sixteen', 17 : 'seventeen', 18 : 'eighteen',
          19 : 'nineteen', 20 : 'twenty',
          30 : 'thirty', 40 : 'forty', 50 : 'fifty', 60 : 'sixty',
          70 : 'seventy', 80 : 'eighty', 90 : 'ninety' }
    k = 1000
    m = k * 1000
    b = m * 1000
    t = b * 1000

    if (num < 20):
        return d[num]

    if (num < 100):
        if num % 10 == 0:
            return d[num]
        else:
            return d[num // 10 * 10] + '-' + d[num % 10]

    if (num < k):
        if num % 100 == 0:
            return d[num // 100] + ' hundred'
        else:
            return d[num // 100] + ' hundred and ' + int_to_en(num % 100)

    if (num < m):
        if num % k == 0:
            return int_to_en(num // k) + ' thousand'
        else:
            return int_to_en(num // k) + ' thousand, ' + int_to_en(num % k)

    if (num < b):
        if (num % m) == 0:
            return int_to_en(num // m) + ' million'
        else:
            return int_to_en(num // m) + ' million, ' + int_to_en(num % m)

    if (num < t):
        if (num % b) == 0:
            return int_to_en(num // b) + ' billion'
        else:
            return int_to_en(num // b) + ' billion, ' + int_to_en(num % b)

    if (num >= t):
        if (num % t == 0):
            return int_to_en(num // t) + ' trillion'
        else:
            return int_to_en(num // t) + ' trillion, ' + int_to_en(num % t)
num=int(input("Enter any positive integer: "))
print(int_to_en(num))
```

```
Enter any positive integer: 3669478521369427
three thousand, six hundred and sixty-nine trillion, four hundred and seventy-eight billion, five hundred and twenty-on
e million, three hundred and sixty-nine thousand, four hundred and twenty-seven
```

Write a Python program to count the number ofstrings where the string length is 2 or more and the first and last character are same from a given list of strings.

Example: Input: ['abc', 'xyz', 'aba', '1221']

Output: 2

In [10]:
```
def match_words(words):
  ctr = 0

  for word in words:
    if len(word) > 1 and word[0] == word[-1]:
      ctr += 1
  return ctr

print(match_words(['abc', 'xyz', 'aba', '1221']))
```

```
2
```

Find the indices of all occurrences of target in the uneven matrix

An irregular/uneven matrix, or ragged matrix, is a matrix that has a different number of elements in each row. Ragged matrices are not used in linear algebra, since standard matrix transformations cannot be performed on them, but they are useful as arrays in computing.

Write a Python program to find the indices of all occurrences of target in the uneven matrix.

Input: [[1, 3, 2, 32, 19], [19, 2, 48, 19], [], [9, 35, 4], [3, 19], 19]

Output: [[0, 4], [1, 0], [1, 3], [4, 1]]

Input: [[1, 2, 3, 2], [], [7, 9, 2, 1, 4],2]

Output: [[0, 1], [0, 3], [2, 2]]

In [14]:
```
def test(M, T):
    L=[]
    for i,row in enumerate(M):
        for j,k in enumerate(row):
            if k==T:
                L.append([i,j])
    return L
M = [[1, 3, 2, 32, 19], [19, 2, 48, 19], [], [9, 35, 4], [3, 19]]
T = 19
print("Matrix:")
print(M)
print("Target value:")
print(T)
print("Indices of all occurrences of the target value in the said uneven matrix:")
print(test(M,T))
```

```
Matrix:
[[1, 3, 2, 32, 19], [19, 2, 48, 19], [], [9, 35, 4], [3, 19]]
Target value:
19
Indices of all occurrences of the target value in the said uneven matrix:
[[0, 4], [1, 0], [1, 3], [4, 1]]
```

Write a Python program to find the numbers that are greater than 10 and have odd first and last digits.

Input: [1, 3, 79, 10, 4, 1, 39, 62]

Output: [79, 39]

Input: [11, 31, 77, 93, 48, 1, 57]

Output: [11, 31, 77, 93, 57]

In [19]:
```
def test(nums):
    L=[]
    for i in nums:
        if i>10:
            if int(str(i)[len(str(i))-1])%2==1 and int(str(i)[0]) % 2==1:
                L.append(i)
    return L

nums = [1, 3, 79, 10, 4, 1, 39]
print("Original list of numbers:")
print(nums)
print("Numbers of the said array that are greater than 10 and have odd first and last digits:")
print(test(nums))
```

```
Original list of numbers:
[1, 3, 79, 10, 4, 1, 39]
Numbers of the said array that are greater than 10 and have odd first and last digits:
[79, 39]
```

write a Python program to create a list containing that number in between each pair of adjacent numbers.

Input: [12, -7, 3, -89, 14, 88, -78, -1, 2, 7]

Separator: 6

Output: [12, 6, -7, 6, 3, 6, -89, 6, 14, 6, 88, 6, -78, 6, -1, 6, 2, 6, 7]

Input: [1, 2, 3, 4, 5, 6]

Separator: 9

Output: [1, 9, 2, 9, 3, 9, 4, 9, 5, 9, 6]

In [25]:
```
def test(nums, sep):
    L=[]
    for i in range(len(nums)):
        if i==len(nums)-1:
            L.append(nums[i])
        else:
            L.append(nums[i])
            L.append(sep)
    return L
nums = [12, -7, 3, -89, 14, 88, -78, -1, 2, 7]
separator = 6
print("List of numbers:",nums)
print("Separator:",separator)
print("Inject the separator in between each pair of adjacent numbers of the said list:")
print(test(nums,separator))
```

```
List of numbers: [12, -7, 3, -89, 14, 88, -78, -1, 2, 7]
Separator: 6
Inject the separator in between each pair of adjacent numbers of the said list:
[12, 6, -7, 6, 3, 6, -89, 6, 14, 6, 88, 6, -78, 6, -1, 6, 2, 6, 7]
```

Write a Python program to start with a list of integers, keep every other element in place and otherwise sort the list.

Input: [2, 5, 6, 3, 1, 4, 34]

Output: [1, 5, 2, 3, 6, 4, 34]

Input: [8, 0, 7, 2, 9, 4, 1, 2, 8, 3]

Output: [1, 0, 7, 2, 8, 4, 8, 2, 9, 3]

In [33]:
```
nums=[8, 0, 7, 2, 9, 4, 1, 2, 8, 3]
L1=nums[::2]
L2=nums[1::2]
L1.sort()
L=[]
for i in range(len(L1)):
    if len(L1)==len(L2):
        L.append(L1[i])
        L.append(L2[i])
    else:
        L.append(L1[i])
print(L)
```

```
[1, 0, 7, 2, 8, 4, 8, 2, 9, 3]
```

Write a Python program to get the single digits in numbers sorted backwards and converted to English words.

Input: [1, 3, 4, 5, 11]

Output: ['five', 'four', 'three', 'one']

Input: [27, 3, 8, 5, 1, 31]

Output: ['eight', 'five', 'three', 'one']

In [40]:
```
def test(nums):
    digits = {None: 'zero', 1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five', 6: 'six', 7: 'seven', 8: 'eight', 9: '
    nums.sort()
    L=[digits[i] for i in nums if i>=0 and i<10]
    L.reverse()
    print(L)
nums= [27, 3, 8, 5, 1, 31]
test(nums)
```

```
['eight', 'five', 'three', 'one']
```

Write a Python program to find the following strange sort of list of numbers: the first element is the smallest, the second is the largest of the remaining, the third is the smallest of the remaining, the fourth is the smallest of the remaining, etc.

Input: [1, 3, 4, 5, 11]

Output: [1, 11, 3, 5, 4]

Input: [27, 3, 8, 5, 1, 31]

Output: [1, 31, 3, 27, 5, 8]

Input: [1, 2, 7, 3, 4, 5, 6]

Output: [1, 7, 2, 6, 3, 5, 4]

In [44]:
```
nums = eval(input("Enter list: "))
temp=nums.copy()
L=[]
i=0
while i<len(temp):
    if i%2==0:
        L.append(min(nums))
        nums.remove(min(nums))
        i+=1
    else:
        L.append(max(nums))
        nums.remove(max(nums))
        i+=1
print(L)
```

```
Enter list: [27, 3, 8, 5, 1, 31]
[1, 31, 3, 27, 5, 8]
```

Write a Python program to find four positive even integers whose sum is a given

integer.

Input: n = 100

Output: [94, 2, 2, 2]

Input: n = 1000

Output: [994, 2, 2, 2]

Input: n = 10000

Output: [9994, 2, 2, 2]

Input: n = 1234567890

In [49]:
```python
def test(n):
    for a in range(n, 0, -1):
        if a % 2 != 0:
            continue
        for b in range(n - a, 0, -1):
            if b % 2 != 0:
                continue
            for c in range(n - b - a, 0, -1):
                if c % 2 != 0:
                    continue
                for d in range(n - b - c - a, 0, -1):
                    if d % 2 != 0:
                        continue
                    if a + b + c + d == n:
                        return [a, b, c, d]

n = int(input("Enter a number"))
print("Four positive even integers whose sum is",n)
print(test(n))
```

```
Enter a number1234567890
Four positive even integers whose sum is 1234567890
[1234567884, 2, 2, 2]
```

Write a python program to implement linear search.

Linear search is a method of finding elements within a list. It is also called a sequential search. It is the simplest searching algorithm because it searches the desired element in a sequential manner. It compares each and every element with the value that we are searching for. If both are matched, the element is found, and the algorithm returns the key's index position.

In [50]:
```
def linearsearch(arr, x):
   for i in range(len(arr)):
      if arr[i] == x:
         return i
   return -1
arr = ['t','u','t','o','r','i','a','l']
x = 'a'
print("element found at index "+str(linearsearch(arr,x)))
```

```
element found at index 6
```

Find three numbers from an array such that the sum of three numbers equal to zero

In [51]:
```
# return a list of lists of length 3
def three_Sum(num):
    if len(num)<3:
        return []
    num.sort()
    result=[]
    for i in range(len(num)-2):
        left=i+1
        right=len(num)-1
        if i!=0 and num[i]==num[i-1]:
            continue
        while left<right:
            if num[left]+num[right]==-num[i]:
                result.append([num[i],num[left],num[right]])
                left=left+1
                right=right-1
                while num[left]==num[left-1] and left<right:left=left+1
                while num[right]==num[right+1] and left<right: right=right-1
            elif num[left]+num[right]<-num[i]:
                left=left+1
            else:
                right=right-1
    return result

nums1=[-1,0,1,2,-1,-4]
nums2 = [-25,-10,-7,-3,2,4,8,10]
print(three_Sum(nums1))
print(three_Sum(nums2))
```

```
[[-1, -1, 2], [-1, 0, 1]]
[[-10, 2, 8], [-7, -3, 10]]
```

Write a python program to check a sequence of numbers is a geometric progression or not.

In [53]:
```
def is_geometric(li):
    if len(li) <= 1:
        return True
    # Calculate ratio
    ratio = li[1]/(li[0])
    # Check the ratio of the remaining
    for i in range(1, len(li)):
        if li[i]/(li[i-1]) != ratio:
            return False
    return True

print(is_geometric([2, 6, 18, 54]))

print(is_geometric([10, 5, 2.5, 1.25]))

print(is_geometric([5, 8, 9, 11]))
```

```
True
True
False
```

In [4]:
```
def mirrorChars(input,k):
    # create dictionary
    original = 'abcdefghijklmnopqrstuvwxyz'
    reverse = 'zyxwvutsrqponmlkjihgfedcba'
    dictChars = {}
    for i in range(len(original)):
        dictChars[original[i]]=reverse[i]

    # separate out string after length k to change
    # characters in mirror
    prefix = input[0:k-1]
    suffix = input[k-1:]
    mirror = ''
    # change into mirror
    for i in range(0,len(suffix)):
        mirror = mirror + dictChars[suffix[i]]
    # concat prefix and mirrored part
    print (prefix+mirror)

input = 'ljietengx'
k = 3
mirrorChars(input,k)
```

```
ljrvgvmtc
```

In [5]:
```
test_dict = {'Gfg' : {"a" : 7, "b" : 9, "c" : 12},
'is' : {"a" : 15, "b" : 19, "c" : 20},
'best' :{"a" : 5, "b" : 10, "c" : 2}}
# printing original dictionary
print("The original dictionary is : " + str(test_dict))
# initializing key
temp = "c"
# using keys() and values() to extract values
res = [sub[temp] for sub in test_dict.values() if temp in sub.keys()]
# printing result
print("The extracted values : " + str(res))
```

```
The original dictionary is : {'Gfg': {'a': 7, 'b': 9, 'c': 12}, 'is': {'a': 15, 'b': 19, 'c': 20}, 'best': {'a': 5,
'b': 10, 'c': 2}}
The extracted values : [12, 20, 2]
```

In [7]:
```
def Convert(tup, di):
    for a, b in tup:
        di.setdefault(a, []).append(b)
    return di
tups = [("akash", 10), ("gaurav", 12), ("anand", 14),
("suraj", 20), ("akhil", 25), ("ashish", 30)]
dictionary = {}
print (Convert(tups, dictionary))
```

```
{'akash': [10], 'gaurav': [12], 'anand': [14], 'suraj': [20], 'akhil': [25], 'ashish': [30]}
```

"""Write a Python program to manage and analyze a small library's book database using strings, lists, and dictionaries. Static Input: Use the following list of strings as input data (no user input required):

books_data = ["Inferno,Dan Brown,Thriller,450,4.5", "Atomic Habits,James Clear,SelfHelp,550,4.8", "Ikigai,Hector Garcia,SelfHelp,300,4.2", "Dune,Frank Herbert,ScienceFiction,600,4.6", "1984,George Orwell,ScienceFiction,500,4.7"]

Your Task: Convert the above list of strings into a dictionary, where: The key = book title The value = another dictionary with: {"Author": ..., "Genre": ..., "Price": ..., "Rating": ...}

Perform the following analyses: a. Display the average price of all books. b. Display the highest-rated book (title, author, rating). c. Group books by genre and show the count of books in each genre. d. Display all ScienceFiction books sorted by rating (descending). e. (Bonus) Display the most expensive book in each genre.

Expected Output:

Average Price of all books: 480.0

Highest Rated Book: Atomic Habits by James Clear (Rating: 4.8) Books by Genre: Thriller: 1 SelfHelp: 2 ScienceFiction: 2 ScienceFiction Books (sorted by rating):

1. 1984 — 4.7
2. Dune — 4.6 Most Expensive Book per Genre: Thriller: Inferno (450) SelfHelp: Atomic Habits (550) ScienceFiction: Dune (600)"""

In [1]:
```
# Static input
books_data = [
    "Inferno,Dan Brown,Thriller,450,4.5",
    "Atomic Habits,James Clear,SelfHelp,550,4.8",
    "Ikigai,Hector Garcia,SelfHelp,300,4.2",
    "Dune,Frank Herbert,ScienceFiction,600,4.6",
    "1984,George Orwell,ScienceFiction,500,4.7"
]

# Step 1: Convert list of strings into a nested dictionary
books_dict = {}

for record in books_data:
    parts = record.split(",")  # Split each string by comma
    title = parts[0].strip()
    author = parts[1].strip()
    genre = parts[2].strip()
    price = float(parts[3].strip())
    rating = float(parts[4].strip())

    books_dict[title] = {
        "Author": author,
        "Genre": genre,
        "Price": price,
        "Rating": rating
    }
print(books_dict)
# Step 2a: Average price of all books
total_price = sum(book["Price"] for book in books_dict.values())
average_price = total_price / len(books_dict)
print(f"Average Price of all books: {average_price:.1f}\n")

# Step 2b: Highest-rated book
highest_rated_book = max(books_dict.items(), key=lambda x: x[1]["Rating"])
title, info = highest_rated_book
print("Highest Rated Book:")
print(f"{title} by {info['Author']} (Rating: {info['Rating']})\n")

# Step 2c: Books grouped by genre
genre_count = {}
for info in books_dict.values():
    genre = info["Genre"]
    genre_count[genre] = genre_count.get(genre, 0) + 1

print("Books by Genre:")
for genre, count in genre_count.items():
    print(f"{genre}: {count}")
print()

# Step 2d: ScienceFiction books sorted by rating (descending)
sci_fi_books = [(title, info["Rating"]) for title, info in books_dict.items() if info["Genre"] == "ScienceFiction"]
sci_fi_books.sort(key=lambda x: x[1], reverse=True)

print("ScienceFiction Books (sorted by rating):")
for i, (title, rating) in enumerate(sci_fi_books, start=1):
    print(f"{i}. {title} – {rating}")
print()

# Step 2e: Most expensive book per genre (Bonus)
most_expensive_per_genre = {}

for title, info in books_dict.items():
    genre = info["Genre"]
    price = info["Price"]
    if genre not in most_expensive_per_genre or price > most_expensive_per_genre[genre]["Price"]:
        most_expensive_per_genre[genre] = {"Title": title, "Price": price}

print("Most Expensive Book per Genre:")
for genre, data in most_expensive_per_genre.items():
    print(f"{genre}: {data['Title']} ({data['Price']})")
```

```
{'Inferno': {'Author': 'Dan Brown', 'Genre': 'Thriller', 'Price': 450.0, 'Rating': 4.5}, 'Atomic Habits': {'Author': 'J
ames Clear', 'Genre': 'SelfHelp', 'Price': 550.0, 'Rating': 4.8}, 'Ikigai': {'Author': 'Hector Garcia', 'Genre': 'SelfH
elp', 'Price': 300.0, 'Rating': 4.2}, 'Dune': {'Author': 'Frank Herbert', 'Genre': 'ScienceFiction', 'Price': 600.0, 'R
ating': 4.6}, '1984': {'Author': 'George Orwell', 'Genre': 'ScienceFiction', 'Price': 500.0, 'Rating': 4.7}}
Average Price of all books: 480.0

Highest Rated Book:
Atomic Habits by James Clear (Rating: 4.8)

Books by Genre:
Thriller: 1
SelfHelp: 2
ScienceFiction: 2

ScienceFiction Books (sorted by rating):
1. 1984 – 4.7
2. Dune – 4.6

Most Expensive Book per Genre:
Thriller: Inferno (450.0)
SelfHelp: Atomic Habits (550.0)
ScienceFiction: Dune (600.0)
```

```
In [ ]:
```

```
In [ ]:
```