1. String Rotation

Input: rhdt:246, ghftd:1246

Output: trhd, ftdgh

Explanation:
Here, every string (rhdt : 1246) is associated with a number, separated by semicolon, if sum of square of digit is even the rotate the string right by 1 position. If square of digit is odd the rotate the string left by 2 position.

For first case:
2*2+4*4+6*6=84 which is even so rotate string, rotate right by 1 so "rhdt" will be "trhd"

For second case:
1*1+2*2+4*4+6*6=85 which is odd so rotate string left by 2 so "ghftd" will be "ftdgh"

```
def sumSqrDigit(num):
    X = int(num)
    #rev = 1
    N = 0
    while(X>0):
        rev = X%10
        rev *= rev
        N += rev
        X = X//10
    return N

def rotateRight(string):
    n=''
    x=''
    n+=string[:-1]
    x+=string[-1]
    x+=n
    return x

def rotateLeft(string):
    n=''
    x=''
    n+=string[:2]
    x+=string[2:]
    x+=n
    return x

series = input().split(':')

for i in series:
    if(i.isdigit()):
        n=i
    else:
        stg=i

if(sumSqrDigit(n)%2==0):
    print(rotateRight(stg))
else:
    print(rotateLeft(stg))
```

2. Matrix with Highest Sum

Take string input then form all possible mXm square matrix, and print the matrix with maximum sum.
In case, two or more square matrix has maximum sum then print largest matrix followed by next largest matrix and so on.
In case, more than one matrix has same size print in order of their occurrence.

INPUT: 6, 3, 6, 20, 3, 6,-15, 3, 3

OUTPUT:
6 3 6
20 3 6
-15 3 3

6 3
6 20

6 20
3 6

Explanation:-
6 3 6
20 3 6 -> its sum is 35 and is order 3*3,
-15 3 3

6 3
6 20 -> its sum is 35 and is order 2*2,

6 20
3 6 -> its sum is 35 and is order 2*2,

```
import math

instr=list(map(int, input().split(",")))

m=2

result={}

k=int(math.sqrt(len(instr)))

s1=0
s1_a=[]
L=[]

while(m<=k):
    j=0
    while(j<=len(instr)-(m*m)):
        matrix=[]
        i=j
        new_sum=0
        while(i!=(m*m)+j):
            matrix.append(instr[i:i+m])
            new_sum+=sum(instr[i:i+m])
            i+=m
        #print(matrix)
```

```
        if(new_sum>s1):
            s1=new_sum
            s1_a.append(matrix.copy())
            L.append(m)
        elif(new_sum==s1):
            s1_a.append(matrix.copy())
            L.append(m)
        j+=1
    m+=1

for i in s1_a:
    for j in i:
        for k in j:
            print(" ", end='')
            print(k,end=" ")
        print("")
```

3. Longest Subarray

Input = {3, 5, 8, 2, 19, 12, 7, 11}

One have to find the longest subarray such that its element satisfies the following condition:

x[i]=x[i- 1]+x[i-2]

If more than one subarray of is found of maximum length one has to print the array which starts with the minimum element and if they are also same then the array with minimum second eleme nt and so on.

If no subarray is found one has to print -1.

Output = {2, 5, 7, 12, 19}

```
inList = list(map(int, input().split(',')))

sublist = []
L=0
lsa = []

for i in range(len(inList)):
    count = 0
    l = i
    j = 0
    c=0
    while(j<len(inList)):
        if(l==j or inList[i]>inList[j]):
            j=j+1
        else:
            if(count==0):
                temp = inList[l] + inList[j]
            else:
                temp = sublist[-1][-1] + sublist[-1][-2]
            if(temp<=max(inList) and temp in inList):
                count+=1
                if(count == 1):
                    sublist.append([])
                    sublist[-1].append(inList[i])
                    sublist[-1].append(inList[j])
                    sublist[-1].append(temp)
                elif(count>1):
```

```
                sublist[-1].append(temp)
            else:
                if((len(sublist[-1]))>L):
                    if(len(lsa)):
                        lsa.pop()
                    lsa.append(sublist[-1])
                    L = len(sublist[-1])
                elif((len(sublist[-1]))==L):
                    if(sublist[-1] not in lsa):
                        lsa.append(sublist[-1])
                        L = len(sublist[-1])
                count=0
                l=i
                j+=1

print(min(lsa))
```

4. Matrix with same consecutive number

Given an inmatrix mxn matrix(m,n>3).You have to perform the following operations:
i) Find the same consecutive numbers in the matrix either vertically, horizontally or diagonally such that atleast four consecutive nos. are there.
ii) If there are more than one such set of numbers then print an integer outnum which is the minimum of such consecutive nos.
iii) If no such no. exists print -1

Input: First line contains an integer m.
Second m lines have n space separated integers (note: n was not given)
Output: An integer outnum or -1 if no such nos. exists

5 1
7 8 9 5 4 2
5 7 9 4 5 2
6 8 7 9 2 2
1 4 2 7 6 2
1 1 1 1 1 4

5 -1
1 2 2 2 5 4
1 1 3 5 4 9
1 4 1 8 2 6
2 5 8 7 4 1
2 9 3 3 3 7

```
m = int(input())

if m<4:
    sys.exit("Wrong Input")

matrix = []

n=0

for i in range(m):
    arr = list(map(int, input().split()))
    if(n<len(arr)):
        n=len(arr)
```

```
    matrix.append(arr[:n])
    #print(matrix[i])

countV=1
countH=1
countD=1

count=[]
diag = []

for j in range(m):
    l=0
    while(l<(m-1)):
        if(matrix[l][j]==matrix[l+1][j]):
            countV+=1
            if(countV==4):
                count.append(matrix[l][j])
        else:
            countV=1
        l+=1
    for k in range(n-1):
        if(matrix[j][k]==matrix[j][k+1]):
            countH+=1
            if(countH==4):
                count.append(matrix[j][k])
        else:
            countH=1
        if(j==k):
            diag.append(matrix[i][j])

D = len(diag)

for x in range(D-1):
    if(diag[x]==diag[x+1]):
        countD+=1
        if(countD==4):
            count.append(diag[x])
    else:
        countD=1

if(count == []):
    print('-1')
else:
    print(min(count))
```

5. Largest Possible Even No.

Form the Largest Possible Even Number from the given Alphanumeric String after removing all duplicate digits.

If No even number can form print -1

Case 1:

Input : Infosys@337

Output : -1

Case 2:
Input : Hello#81@21349
Output : 984312

```
inStr = input()

digi = '1234567890'
digit = []
k=0
f=0
l=0
X=0
flag=0
out=''

for i in inStr:
    if(i in digi):
        if(i in digit):
            continue
        else:
            digit.append(i);
        if(int(i)%2==0):
            flag = 1;

if(flag == 0):
    print(-1)

else:
    N = len(digit)
    digit.sort()
    while(int(k)<1):
        if(f<1 and int(digit[l])%2==0):
            k=digit[l]
            f=1
        l+=1
    for j in range(N):
        X = (N-j)-1
        if(k != digit[X]):
            out+=digit[X]
    out+=k

    print(out)
```

6. Special Character Sum – Even/Odd

1. If Even Special Character find in inStr, outStr contains digits from inStr starting from Even then Odd then even-odd so on.

2. If Odd Special Character find in inStr, outStr contains digits from inStr starting from Odd then Even then odd-even so on.

3. If there any numbers (additional) left append them at last

Case 1 :
inStr = t9@a42g&516
outStr = 492561

Case 2 :
inStr = 5u6@n25g7#@
outStr = 56527

```
inStr = input()
special = '!@#$&'
digit = '1234567890'
x = 0
y = 0
even = []
odd = []

for i in inStr:
    if(i in special):
        x+=1
    elif(i in digit):
        if(int(i)%2==0):
            even.append(i)
        else:
            odd.append(i)

if(len(even)<=len(odd)):
    N = len(even)
    flag=1
elif(len(even)>=len(odd)):
    N = len(odd)
    flag=2

out = ''

if(x%2 == 0):
    for j in range(N):
        out+=even[j]
        out+=odd[j]
    if(flag == 1):
        y = j+1
        while(y<len(odd)):
            out+=odd[y]
            y += 1
    if(flag == 2):
        y = j+1
        while(y<len(even)):
            out+=even[y]
            y += 1

else:
    for k in range(N):
        out+=odd[k]
        out+=even[k]
    if(flag == 2):
        y = k+1
        while(y<len(even)):
            out+=even[y]
```

```
        y += 1
    if(flag == 1):
        y = k+1
        while(y<len(odd)):
            out+=odd[y]
            y += 1

print(out)
```

7. String contains 5 and 8

Input:
A string of comma separated numbers, the numbers 5 and 8 are present in the List. (8 always comes after 5)

Problem:
Num1: Add all numbers which do not lie between 5 and 8 (excluding 5,8)
Num2: Number formed by concatenating all numbers from 5 to 8 (Including 5,8)
Output: Num1 + Num2

Case 1:
Input = 3,2,6,5,1,4,8,9
Output = 5168

(Num1 = 3+2+6+9 = 20
Num2 = '5'+'1'+'4'+'8' = 5148
Output = 5148 + 20 = 5168)

Case 2:
Input = 3,1,5,8
Output = 62

```
inList = input().split(',')

five = inList.index('5')
eight = inList.index('8')
num1 = 0
num2 = ''

for i in range(len(inList)):
    if(i<five):
        num1 += int(inList[i])
    if(i>five and i>eight):
        num1 += int(inList[i])

    elif(i>=five and i<=eight):
        num2 += inList[i]

out = int(num1)+int(num2)

print(out)
```

8. Parenthesis Problem

A non empty string instr containing only parenthesis (,),{,},[,]. It returns outstr based on following-

- **instr is properly nested and return 0 (zero).**
- **instr not properly nested, returns position of element in instr.**
- **position starts from 1.**

Test cases:

input	**output**
{([])}[]	**0**
([)()]	**3**
[[()]	**6 (n+1 for lat element i.e. 5+1 =6)**

```
openList = ['[','{','(']
closeList = [']','}',')']

def balance(myStr):
    stack= []
    c=0
    for i in myStr:
        c+=1
        if i in openList:
            stack.append(i)
        elif i in closeList:
            pos = closeList.index(i)
            if ((len(stack) > 0) and (openList[pos] == stack[len(stack)-1])):
                stack.pop()
            else:
                return c
    if (len(stack) == 0):
        return 0
    else:
        n = len(myStr)
        return n+1

instr = input()

print(balance(instr))
```

9. Longest Substring which is unique

A string is given, we have to find the longet substring which is unique(that has no repetation) and has a minimum size of 3. If more than one substring is found with max length then we have to print the one which appeared first in the string.If no substring is present which matches the condition then we have to print -1.

input - "A@bcd1abx"
output - "A@bcd1"

Note:
" A@bcd1a" is not a unique substring as it contains "A" and "a" and substring "bcd1a" does not appear first.

10. Longest Prefix which is also suffix

A non-empty string instr containing only alphabets.
Print length of longest prefix in instr which is same as suffix. prefix and suffix should not overlap in instr.

Print -1 if no prefix exists which is also the suffix without overlap.
Do case sensitive comparison wherever necessary.
position starts from 1.

Test cases:

Input	output
xxAbcxxAbcxx	2
Racecar	-1

11. Reverse String keeping special character at same place

Special string reverse

Input Format: b@rd

Output Format: d@rb

Explanation:

We should reverse the alphabets of the string by keeping the special characters in the same position

12. OTP Generation

Input Format:

13456

Output Format:

1925

Explanation:

Take the string of numbers and generate a four digit OTP such that

1. If the number is odd square it.
2. If the number is even ignore it.

13. Longest substring with common letter from 2 string

S1="staobplk"
S2="tsodpq"
1. Identify common letters in both strings
2. Form all possible substrings possible with those letters
--> substrings should be formed in such a way that...
--->Common letters should be taken first occurrence of string 2,
---->substring should be formed by sequences common letters

A. Out of all possible substrings Print longest substring
B. If >1 substring has same highest length then print the substring with letters first occurred in str2 and if no common letters in 2 strings print "X" Capital X

S1="staobplk"
S2="tsodpq"

Out = 'top'

14. Pronic Number

Input1: 93012630
Output2: 2,6,12,30,930
We should divide the total number into substrings and we should verify each num is pronic num or not if pronic we should print that num
Pronic: means it is a multiple of two consecutive integers
Ex: 6->2*3 it's a pronic
12->3*4 it's a pronic

Input2: 12665042
Output2: 2,6,12,42,650

15. **Parking Slot Problem**
16. **N swap minimum No.**
17. **Generate Password**
18. **Longest Substring Palindrome**
19. **Nearest Palindrome**