1. Create a function that takes a list and returns a new list containing only prime numbers.

**Examples**

filter\_primes([7, 9, 3, 9, 10, 11, 27]) ➞ [7, 3, 11]

filter\_primes([10007, 1009, 1007, 27, 147, 77, 1001, 70]) ➞ [10007, 1009]

filter\_primes([1009, 10, 10, 10, 3, 33, 9, 4, 1, 61, 63, 69, 1087, 1091, 1093, 1097]) ➞ [1009, 3, 61, 1087, 1091, 1093, 1097]

**Ans:**

def filter\_primes(lst):

l=[]

for i in lst:

if i==1:

continue

found = True

for j in range(2, (i//2)+1):

if i%j==0:

found=False

if found:

l.append(i)

print(l)

2. Once a water balloon pops, is soaks the area around it. The ground gets drier the further away you travel from the balloon.

The effect of a water balloon popping can be modeled using a list. Create a function that takes a list which takes the pre-pop state and returns the state after the balloon is popped. The pre-pop state will contain at most a single balloon, whose size is represented by the only non-zero element.

**Examples**

pop([0, 0, 0, 0, 4, 0, 0, 0, 0]) ➞ [0, 1, 2, 3, 4, 3, 2, 1, 0]

pop([0, 0, 0, 3, 0, 0, 0]) ➞ [0, 1, 2, 3, 2, 1, 0]

pop([0, 0, 2, 0, 0]) ➞ [0, 1, 2, 1, 0]

pop([0]) ➞ [0]

**Ans:**

def pop(lst):

l=[]

count=0

found=False

for i in lst:

if i != 0:

l.append(i)

found=True

count-=1

continue

l.append(count)

if found:

count-=1

else:

count+=1

print(l)

3. "Loves me, loves me not" is a traditional game in which a person plucks off all the petals of a flower one by one, saying the phrase "Loves me" and "Loves me not" when determining whether the one that they love, loves them back.

Given a number of petals, return a string which repeats the phrases "Loves me" and "Loves me not" for every alternating petal, and return the last phrase in all caps. Remember to put a comma and space between phrases.

**Examples**

loves\_me(3) ➞ "Loves me, Loves me not, LOVES ME"

loves\_me(6) ➞ "Loves me, Loves me not, Loves me, Loves me not, Loves me, LOVES ME NOT"

loves\_me(1) ➞ "LOVES ME"

**Ans:**

def loves\_me(n):

odd=",Loves me"

even=",Loves me not"

string=""

for i in range(1, n+1):

if i==n and i%2==0:

string+=even.upper()

if i==n and i%2!=0:

string+=odd.upper()

if i%2==0 and i!=n:

string+=even

if i%2!=0 and i!=n:

string+=odd

print(string[1:])

4. Write a function that sorts each string in a list by the letter in alphabetic ascending order (a-z).

**Examples**

sort\_by\_letter(["932c", "832u32", "2344b"])

➞ ["2344b", "932c", "832u32"]

sort\_by\_letter(["99a", "78b", "c2345", "11d"])

➞ ["99a", "78b", "c2345", "11d"]

sort\_by\_letter(["572z", "5y5", "304q2"])

➞ ["304q2", "5y5", "572z"]

sort\_by\_letter([])

➞ []

**Ans:**

def sort\_by\_letter(lst):

dic={}

output=[]

for l in lst:

for i in l:

if i.isalpha():

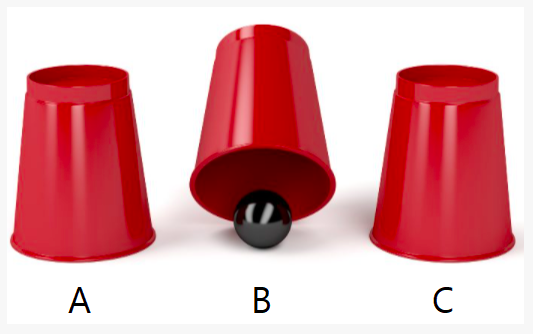
dic[i]=l

for i in sorted(dic.keys()):

output.append(dic[i])

print(output)

5. There are three cups on a table, at positions A, B, and C. At the start, there is a ball hidden under the cup at position B.



However, I perform several swaps on the cups, which is notated as two letters. For example, if I swap the cups at positions A and B, I could notate this as AB or BA.

Create a function that returns the letter position that the ball is at, once I finish swapping the cups. The swaps will be given to you as a list.

**Example**

cup\_swapping(["AB", "CA", "AB"]) ➞ "C"

# Ball begins at position B.

# Cups A and B swap, so the ball is at position A.

# Cups C and A swap, so the ball is at position C.

# Cups A and B swap, but the ball is at position C, so it doesn't move.

**Ans:**

def cup\_swapping(lst):

string="ABC"

for l in lst:

string=string.replace(l[1], "\*")

string=string.replace(l[0], l[1])

string=string.replace("\*", l[0])

print(string[1])