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:

In [15]:

```python
def filter_primes(in_list):
    output = []
    for i in in_list:
        if i==2 or i==3:
            output.append(i)
        elif ((i+1)%6 ==0) or ((i-1)%6 == 0) and i!=1:
            output.append(i)
    temp = output.copy()
    for i in temp:
        for n in range(2,i):
            if i%n == 0:
                output.remove(i)
                break
    print(f'filter_primes({in_list}) → {output}')
filter_primes([7, 9, 3, 9, 10, 11, 27])
filter_primes([10007, 1009, 1007, 27, 147, 77, 1001, 70])
filter_primes([1009, 10, 10, 10, 3, 33, 9, 4, 1, 61, 63, 69, 1087, 1091, 1093, 1097])
```

```
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]
```

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:

In [1]:

```
def pop(in_list):
    num = [i for i in in_list if i!=0 ]
    pos = 0
    for i in range(len(in_list)):
            if in_list[i]!=0:
                pos = i
    output = []
    if in_list==[0]:
        output = [0]
    else:
        for i in range(0,pos+1):
            output.append(i)
        for i in range(pos-1,0,-1):
            output.append(i)
        output.append(0)
    print(f'pop({in_list}) →{output}')
pop([0, 0, 0, 0, 4, 0, 0, 0, 0])
pop([0, 0, 0, 3, 0, 0, 0])
pop([0, 0, 2, 0, 0])
pop([0])
```

```
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]
```

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:

In [6]:

```
def loves_me(num):
    string = []
    for i in range(num):
        if i%2!=0:
            string.append('Loves me')
        else:
            string.append('Loves me not')
    string[-1] = string[-1].upper()
    print(f'loves_me({num}) → {", ".join(string)}')
loves_me(3)
loves_me(6)
loves_me(1)
```

```
loves_me(3) → Loves me not, Loves me, LOVES ME NOT
loves_me(6) → Loves me not, Loves me, Loves me not, Loves me, Loves me not, LOVES ME
loves_me(1) → LOVES ME NOT
```

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

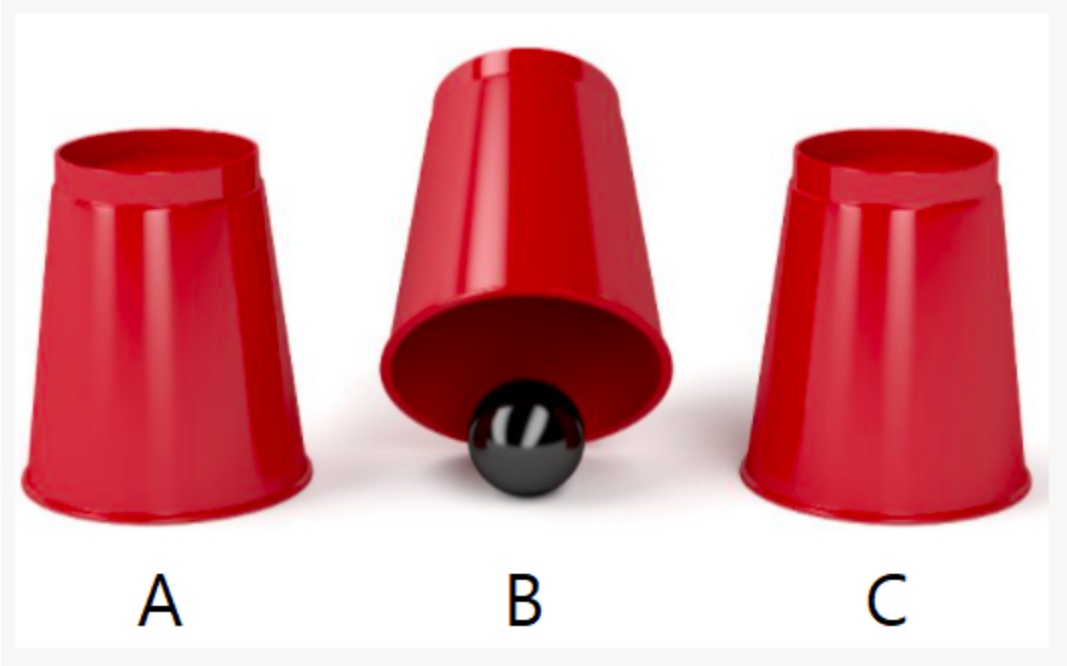
Examples:
sort_by_letter(["932c", "832u32", "2344b"]) ➞ ["2344b", "932c", "832u32"]
sort_by_letter(["99a", "78b", "c2345", "11d"]) ➞ ["99a", "78b", "c2345", "11d"]
sort_by_letter(["572z", "5y5", "304q2"]) ➞ ["304q2", "5y5", "572z"]
sort_by_letter([]) ➞ []

Ans:

In [9]:
```
def sort_by_letter(in_list):
    output = []
    temp = []
    for word in in_list:
        for char in word:
            if char.isalpha():
                temp.append(char)
    temp.sort()
    for char in temp:
        for word in in_list:
            if char in word:
                output.append(word)
    print(f'sort_by_letter({in_list}) ➞ {output}')
sort_by_letter(["932c", "832u32", "2344b"])
sort_by_letter(["99a", "78b", "c2345", "11d"])
sort_by_letter(["572z", "5y5", "304q2"])
```

```
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']
```

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.

Examples:
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:

In [11]:
```
def cup_swapping(in_list):
    pos = 'B'
    for i in in_list:
        if pos in i:
            pos = i.replace(pos,'')
    print(f'cup_swapping({in_list}) ➞ {pos}')
cup_swapping(["AB", "CA", "AB"])
```

```
cup_swapping(['AB', 'CA', 'AB']) ➞ C
```