**ASSIGNMENT-11**

**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]

In [1]:

**def** filter\_primes(in\_list):

out\_list **=** []

**for** ele **in** in\_list:

**if** ele **in** [2,3]:

out\_list**.**append(ele)

**elif** ((ele**+**1)**%6** == 0) or ((ele-1)%6 == 0) and ele != 1:

out\_list**.**append(ele)

temp **=** out\_list**.**copy()

**for** ele **in** temp:

**for** num **in** range(2,ele):

**if** ele**%num** == 0:

out\_list**.**remove(ele)

**break**

print(f'filter\_primes({in\_list}) ➞ {out\_list}')

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]

In [2]:

**def** pop(in\_list):

pop\_number **=** sorted(in\_list,reverse**=True**)[0]

output **=** []

**if** pop\_number **==** 0:

output**.**append(0)

**else**:

output**.**extend([x **for** x **in** range((len(in\_list)**-**1)**//**2)])

output**.**append(pop\_number)

output**.**extend([((len(in\_list)**-**2)**//**2)**-**x **for** x **in** range((len(in\_list)**-**1)**//**2)])

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"

In [3]:

**def** loves\_me(in\_num):

out\_string **=** []

**for** ele **in** range(in\_num):

**if** ele**%2** ==0:

out\_string**.**append('Loves me')

**else**:

out\_string**.**append('Loves me not')

out\_string[**-**1] **=** out\_string[**-**1]**.**upper()

print(f'loves\_me({in\_num}) ➞ {", "**.**join(out\_string)}')

loves\_me(3)

loves\_me(6)

loves\_me(1)

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

**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([]) ➞ []

In [4]:

**def** sort\_by\_letter(in\_list):

temp\_list **=** []

output **=** []

**for** ele **in** in\_list:

**for** char **in** ele:

**if** char**.**isalpha():

temp\_list**.**append(char)

temp\_list**.**sort()

**for** char **in** temp\_list:

**for** ele **in** in\_list:

**if** char **in** ele:

output**.**append(ele)

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([])

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([]) ➞ []

**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.

In [5]:

**def** cup\_swapping(swap\_list,intitial\_ball\_pos):

ball\_position **=** intitial\_ball\_pos

**for** ele **in** swap\_list:

**if** ball\_position **in** ele:

ball\_position **=** ele**.**replace(ball\_position,'')

print(f'cup\_swapping({swap\_list}) ➞ {ball\_position}')

cup\_swapping(["AB", "CA", "AB"],'B')

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

cup\_swapping(["AC", "BC", "AB"],'A')

cup\_swapping(['AB', 'CA', 'AB']) ➞ C

cup\_swapping(['AB', 'CA', 'AB']) ➞ B

cup\_swapping(['AC', 'BC', 'AB']) ➞ A