Registration id - SIRSS1066

Name - Himanshu Monat

College / Organization - MIT, Ujjain / Xoriant Solutions

Subject - Assignment 5

Q1. Given a list of integers, write a function to return the sum of all prime numbers in that list.

In [15]:
```
def is_prime(num):
    """Function checks if the given number is prime or not?"""
    for i in range(2, int(num ** 0.5)+1):
        if num % i == 0:
            return False

    return True


def sum_prime(val):
    """Function calculates sum of all the prime numbers present in the list"""
    sum_1 = 0
    obj1 = filter(is_prime, val)
    prime_bool_list = list(obj1)
    for i in range(len(prime_bool_list)):
        sum_1 += prime_bool_list[i]
    return sum_1

li = [2,5,73,6,38,84,3,78]
result = sum_prime(li)
print(result)
```

```
83
```

Q2. Given a list of integers, write a function to check whether the list is strictly increasing or not.

In [11]:
```
def is_increasing_list(li):
    if len(li) != len(set(li)):
        return False

    ordered_li = sorted(li)
    if (li == ordered_li):
        return True
    else:
        return False

li =  [10, 12, 23, 34, 55]
is_increasing_list(li)
```

Out[11]:
```
True
```

Q3. Write a function to check whether a given list is expanding or not (the difference between adjacent elements should keep on increasing).

In [21]:
```
def is_expanding_list(li):
    if len(li) != len(set(li)):
        return False

    diff = 0
    diff1 = 0
    for i in range(1, len(li)):
        diff = li[i] - li[i-1]
        if diff <= diff1:
            return False
        diff1 = diff

    return True

li =  [10, 12, 23, 35, 55]
is_expanding_list(li)
```

Out[21]:
```
True
```

Q4. Write a function to calculate all permutations of a given string. (Without using itertools)

In [104…
```
def permutations(string, step = 0):

    # if we've gotten to the end, print the permutation
    if step == len(string):
        print("".join(string))

    # everything to the right of step has not been swapped yet
    for i in range(step, len(string)):

        # copy the string (store as array)
        string_copy = [character for character in string]

        # swap the current index with the step
        string_copy[step], string_copy[i] = string_copy[i], string_copy[step]

        # recurse on the portion of the string that has not been swapped yet (now it's index will begin with step + 1)
        permutations(string_copy, step + 1)


permutations(str(input()))
```

```
ABC
ABC
ACB
BAC
BCA
CBA
CAB
```