Q1)

Write a Function that inputs a number & prints the multiplication table of number

In [48]:

```
def print_multiplication_table(n: int):
    for i in range(1, 11):
        print(f'{n} * {i} = {n*i}')

print_multiplication_table(2)
```

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

Q2)

Write a Program to print twin primes less than 1000. Twin Primes : If 2 consecutive odd numbers are both primes then they are known as Twin Primes

In [49]:

```
import math

def isPrime(n):
    # Square Root Range
    for i in range(2, math.isqrt(n) + 1):
        if n % i == 0:
            return False
    return True

def twin_prime(n):
    '''2 consecutive odd prime num '''
    ans = [2, 3]
    last_prime = -1
    for i in range(5, n, 2):
        if isPrime(i):
            if last_prime != -1:   # last prime needs pair
                ans.extend((last_prime, i))
                last_prime = -1
            elif ans[-1] == i-2:  # last prime exists in list
                ans.append(i)
                last_prime = -1
            else:
                # this can be first number
                last_prime = i
        else:
            last_prime = -1
    return ans

print(twin_prime(1000))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 29, 31, 41, 43, 59, 61, 71, 73, 101, 103, 107, 10
9, 137, 139, 149, 151, 179, 181, 191, 193, 197, 199, 227, 229, 239, 241, 269,
271, 281, 283, 311, 313, 347, 349, 419, 421, 431, 433, 461, 463, 521, 523, 56
9, 571, 599, 601, 617, 619, 641, 643, 659, 661, 809, 811, 821, 823, 827, 829,
857, 859, 881, 883]
```

Q3)

Write a Program to Find out Prime Factors of Number. Example: prime factors of 56 -

2,2,2,7

In [50]:

```
import math
from collections import defaultdict, Counter

def prime_factors(n):
    if n <= 1:
        return None

    d = defaultdict(int)

    # for all even numbers
    while n % 2 == 0:
        d[2] += 1
        n = n // 2

    # for all odd numbers
    i = 3
    while n != 1:  # till we dont get entire number broken to single unit, di
        while n % i == 0:
            d[i] += 1
            n = n // i
        i += 2

    return d  # Prime Factors with their Counts/Frequency

n = 56
*primeFactors, = Counter(prime_factors(n)).elements()
print(f'{n} :- {primeFactors}')
```

```
56 :- [2, 2, 2, 7]
```

Q4) :

Write a Program to Print Formula of Permutation & Combinations.

Number of Permutations of n objects taken r at a time: p(n,r) = n!/(n-r)!

Number of Combinations of n objects taken r at a time: c(n,r) = n!/(r! * (n-r)!) = p(n,r)/r!

In [51]:

```
def fact(n):
    ''' calculating factorial using non-tailed recursion '''
    if n == 1:
        return 1
    if n == 2:
        return 2
    return n * fact(n-1)

def permutation(n, r):
    '''
        p(n, r) = n! / (n-r)!
    '''
    return fact(n) / fact(n-r)

def combinations(n, r):
    '''
        c(n, r) = n! / (r! * (n-r)!)
                = p(n,r) / r!
    '''
    return permutation(n, r) / fact(r)

n = 10
r = 3
print(f'{permutation(n, r)=}')
print(f'{combinations(n, r)=}')
```

```
permutation(n, r)=720.0
combinations(n, r)=120.0
```

***UTILS* --->**

In [52]:

```
def get_digits(n, func=None, base=10):
    '''
    :param n: number in decimal base 10
    :param func: func to apply to each extracted digit based on {base}
    :param base: base that needs to be considered whilst extracting digits
    return generator of digits
    if func provided then digit is mapped to that func & then returned
    '''
    while n:
        n, r = divmod(n, base)
        yield func(r) if func else r
```

Q5)

Write a function to convert decimal to binary

In [53]:

```
def decimal_to_binary(n):  # Just traverse in reverse order after doing LCM
    return ''.join(map(str, get_digits(n, base=2)))[::-1]

print(decimal_to_binary(13))
```

```
1101
```

Q6)

Write a function cubeSum() that accepts an integer and returns the sums of cubes of individual digits of that number. Use this function to make printArmstrong() & isArmstrong() to print armstrongs numbers

In [54]:

```
def cubesum(n):
    return sum(get_digits(n, lambda x: x**3))

def isArmStrong(n):
    return n == cubesum(n)

def printArmStrong(n):
    *armStrongNums, =  filter(isArmStrong, range(1, n))
    print(armStrongNums)

printArmStrong(1000)
```

```
[1, 153, 370, 371, 407]
```

Q7)

Write a function prodDigits() that inputs a number & returns the product of digits of a number

In [55]:

```
from math import prod
def prodDigits(n):
    return prod(get_digits(n))

print(prodDigits(23))
```

```
6
```

Q8)

If all the digits of number are multiplied with each other repeating the product, the one digit number obtained at last is called the mmultiplicative digital root of n. The number of times digits need to be multiplied to reach one digit is called the multiplicative persistence of n

Eg 86 -> 48 -> 32 -> 6 (MDR 6, MPersistence 3)
341 -> 12 -> 2. (MDR 2, MPersistence 2)

Using function prodDigits(), write func MDR() & MPersistence() that inputs a number & returns its multiplicative digital root & multiplicative persistence respectively.

In [56]:

```
def MDR(n):
    '''
    all digits of num multiply with each other & repeating phenomennon till 1
    that 1 digit num is called as MDR
    &
    num of times you did phenomenon is called as MPersistence
    86 -> 48 -> 32 -> 6 (MDR = 6, MPersistence = 3)
    341 -> 12 -> 2 (MDR = 2, MPersistence = 2)

    :return : (MDR, MPersistence)
    '''
    m_persistence = 0
    while (n // 10) != 0:
        n = prodDigits(n)
        m_persistence += 1

    return n, m_persistence

mdr, m_per = MDR(86)
print(f'MDR :- {mdr}, M-Persistence :- {m_per}')
```

```
MDR :- 6, M-Persistence :- 3
```

Q9)

Write a function sumPdivisors() that find the sum of proper divisors of a number. Proper divisors of numbers are those numbers by which number is divisible, except the number itself For Eg proper divisors of 36 are 1,2,3,4,6,9,12,18

In [57]:

```
from math import isqrt

def proper_divisors(n):
    if n == 1: return []
    e = isqrt(n)  # go till the sqrt(n)
    ans = [1]
    for i in range(2, e):
        q, r = divmod(n, i)
        if r == 0:
            ans.extend([i, q])

    if e*e == n: # check for sqrt number ie e*e == n (to avoid duplicates it
        ans.append(e)
    return ans

def sumPdivisors(n):
    return sum(proper_divisors(n))

n = 36
```

```
print(f'{proper_divisors(n)= }')
print(f'{sumPdivisors(n)= }')
```

```
proper_divisors(n)= [1, 2, 18, 3, 12, 4, 9, 6]
sumPdivisors(n)= 55
```

Q10)

A number is called perfect number if sum of proper divisors of that number is equal to the number. For eg 28 is perfect number (ie 1 + 2 + 4 + 7 + 14 = 28). Write program to print all perfect number in given range.

In [58]:

```
def is_perfect_num(n):
    return n == sumPdivisors(n)

def all_perfect_num(n):
    return [*filter(is_perfect_num, range(1, n+1))]

print(all_perfect_num(1000))
```

```
[28, 496]
```

Q11)

2 diff numbers are called amicable numbers if the sum of proper divisors of each is equal to the other number. For Example 220 & 284 are amicable numbers.
Sum of Proper divisors of 220 = 1+2+4+5+10+11+20+22+44+55+110 = 284 Sum of proper divisors of 284 = 1+2+4+71+142 = 220 Write a program to print pairs of amicable numbers in the range.

In [59]:

```
from itertools import combinations, starmap, compress

def is_amicable_pair(n1, n2):
    return sumPdivisors(n1) == n2 and sumPdivisors(n2) == n1

def all_amicable_pair(l):
    *pairs, = combinations(l, 2)  # need to convert iterator -> list as pairs
    return [*compress(pairs, starmap(is_amicable_pair, pairs))]

print(all_amicable_pair(range(1,1000)))
```

```
[(220, 284)]
```

Q12)

Write a program that can filter odd nums in list using filter function

In [60]:

```
from operator import methodcaller

def filter_odd_nums(l):
    #return filter(lambda x: x & 1, l)
    return [*filter(methodcaller('__rand__', 1), l)]

print(filter_odd_nums(range(1, 10)))
```

```
[1, 3, 5, 7, 9]
```

Q13)

Write a program that can map nums in list to cube of themselves

In [61]:

```
def cube_all(l):
    #return map(lambda x: x**3, l)
    return map(methodcaller('__pow__', 3), l)

print([*cube_all(range(1, 10))])
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Q14)

Write a program that can map & filter to make a list whose elements are cube of even number in a given list

In [62]:

```
from itertools import filterfalse
def filter_even_nums(l):
    #return filter(lambda x: not(x & 1), l)
    return filterfalse(methodcaller('__rand__', 1), l)

print([*filter_even_nums(range(1, 10))])
```

```
[2, 4, 6, 8]
```