In [3]:

```
from functools import reduce

def multiply(x, y):
    """Multiply two given numbers.

    Args:
        x: int
        y: int
    Returns:
        int
    """
    return x*y

def validatePositiveInteger(num):
    """Utility function to validate +ve integer.

    Args:
        num: int
    Returns:
        Boolean True if valid positive number otherwise Boolean False
    """
    if(type(num) != int):
        print('Given number must be type of integer')
        return False
    elif(num < 1):
        print('Given number should not be zero or -ve integer')
        return False
    else:
        return True

def isPrime(num):
    """Check whether a number is Prime or not.

    Args:
        num: int
    Returns:
```

```
        Boolean True If it's Prime number otherwise Boolean False
    """
    if(num > 0):
        for i in range(1, num):
            if(i > 1 and num % i == 0):
                return False
    return True

def getDigits(num):
    """Get individual digits of a number.

    Args:
        num: int, given number
    Returns:
        List
    """

    if(num < 10):
        return num

    digits = []

    divider = 10

    while(True):
        reminder = num % divider
        digits.append(reminder)
        divisor = int(num/divider)
        num = divisor

        if(num < 10):
            digits.append(num)
            break

    return digits

def getDivisors(num):
    """Get proper divisors of the given numbers.
```

```
    Args:
        num: int, given number
    Returns:
        list
    """
    if(False == validatePositiveInteger(num)):
        return

    divisors = []
    for i in range(1, num):
        if(num % i == 0):
            divisors.append(i)

    return 1 if (len(divisors) <= 1) else divisors

def factorial(num):
    """Get Factorial of a given number.

    Args:
        num: int
    Returns:
        int
    """

    if(1 == num):
        return 1

    return num * factorial(num - 1)
```

In [4]:

```
#1. Write a function that inputs a number and prints the multiplication
 table of that number

def generateTable(num):
    """Generate multiplication table

    Args:
        limit: int, given number
    Returns:
        list
```

```
    """

    if(False == validatePositiveInteger(num)):
        return

    table = []

    for i in range(1, 11):
        table.append(i*num)

    return table

print('************************* Output of question 1 ***************
**********')
print('\n\n')
# Question 1 driver
num = int(input('Enter a valid number'))
print(generateTable(num))
```

```
************************* Output of question 1 ***********************
***



Enter a valid number9
[9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
```

In [5]:

```
# 2. Write a program to print twin primes less than 1000.
# If two consecutive odd numbers are both prime then they are known as
 twin primes

def getTwinPrimes(limit=1001):
    """Calculate Twin Prime numbers upto the given limit.

    Args:
        limit: int, max range
    Returns:
        list of tuples
    """
```

```
    twinPrimes = []
    for i in range(2, limit):
        if(i % 2 != 0 and True == isPrime(i) and True == isPrime(i+2)):
            twinPrime = set()
            if(i not in twinPrime):
                twinPrime.add(i)
            if(i+2 not in twinPrime):
                twinPrime.add(i+2)

            twinPrimes.append(twinPrime)

    return twinPrimes;

print('\n\n')
print('************************** Output of question 2 ****************
**********')
print('\n\n')
# Question 2 driver
twinPrimes = getTwinPrimes()
print(twinPrimes)
print(len(twinPrimes))
```

```
************************** Output of question 2 ************************
***



[{3, 5}, {5, 7}, {11, 13}, {17, 19}, {29, 31}, {41, 43}, {59, 61}, {73,
71}, {101, 103}, {107, 109}, {137, 139}, {149, 151}, {179, 181}, {193,
191}, {197, 199}, {227, 229}, {241, 239}, {269, 271}, {281, 283}, {313,
311}, {347, 349}, {419, 421}, {433, 431}, {461, 463}, {521, 523}, {569,
571}, {601, 599}, {617, 619}, {641, 643}, {659, 661}, {809, 811}, {821,
823}, {827, 829}, {857, 859}, {881, 883}]
35
```

In [6]:
```
#3. Write a program to find out the prime factors of a number. Example:
 prime factors of 56 - 2, 2, 2, 7
```

```
def calculatePrimeFactors(num):
    """Calculate prime factors of a given number.

    Args:
        num: int, given number
    Returns:
        list
    """

    if(False == validatePositiveInteger(num)):
        return

    primeFactors = []

    while(num % 2 == 0):
        primeFactors.append(2)
        num = num/2

    i=3
    while(i<=num):
        if(num % i == 0):
            primeFactors.append(i)
            num = num / i

        i = i+2

    return primeFactors

print('\n\n')
print('************************* Output of question 3 ****************
**********')
print('\n\n')
# Question 3 driver
num = int(input('Enter a valid number'))
print(calculatePrimeFactors(num))
```

```
************************* Output of question 3 ***********************
```

```
***


Enter a valid number56
[2, 2, 2, 7]
```

In [25]:

```python
# 4. Write a program to implement these formulae of permutations and co
mbinations.
# 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 is: c(n, r) =
 n! / (r!*(n-r)!) = p(n,r) / r!
def getNumberOfPermutations(n, r):
    """Get number of permutations of n objects taken r at a time: p(n,
 r) = n! / (n-r)!.

    Args:
        n: int
        r: int
    Returns:
        int
    """
    if(False == validatePositiveInteger(n)):
        return

    if(False == validatePositiveInteger(r)):
        return

    return factorial(n)//factorial(n-r)

def getNumberOfCombinations(n, r):
    """Get number of combinations of n objects taken r at a time is: c
(n, r) = n! / (r!*(n-r)!) = p(n,r) / r!

    Args:
        n: int
        r: int
    Returns:
```

```
        int
    """
    if(False == validatePositiveInteger(n)):
        return

    if(False == validatePositiveInteger(r)):
        return

    return getNumberOfPermutations(n,r)//factorial(r)

print('\n\n')
print('************************* Output of question 4 ****************
**********')
print('\n\n')
# Question 4 driver
n = int(input('Enter n'))
r = int(input('Enter r'))
print(getNumberOfPermutations(n, r))
n = int(input('Enter n'))
r = int(input('Enter r'))
print(getNumberOfCombinations(n, r))
```

```
************************* Output of question 4 ***********************
***



Enter n15
Enter r4
32760
Enter n15
Enter r4
1365
```

In [7]:
```
#5. Write a function that converts a decimal number to binary number

def convertDecimalToBinary(num):
```

```
    """Converts a decimal number to binary.

    Args:
        num: int, given number
    Returns:
        string
    """

    if(False == validatePositiveInteger(num)):
        return

    binaryString = ''
    while(num >= 1):
        if(num % 2 == 0):
            num = int(num/2)
            binaryString += '0'
        elif(1 == num):
            binaryString += '1'
            break
        else:
            num = int(num/2)
            binaryString += '1'

    return binaryString[::-1]

print('\n\n')
print('************************** Output of question 5 ****************
**********')
print('\n\n')
# Question 5 driver
num = int(input('Enter a valid number'))
print(convertDecimalToBinary(num))
```

```
************************** Output of question 5 ***********************
***
```

```
Enter a valid number64
1000000
```

In [10]:

```python
# 6. Write a function cubesum() that accepts an integer and
# returns the sum of the cubes of individual digits of that number.
# Use this function to make functions PrintArmstrong() and isArmstrong
()
# to print Armstrong numbers and to find whether is an Armstrong numbe
r.
def cubesum(num):
    """Get sum of the cubes of individual digits of that number.

    Args:
        int
    Returns
        int
    """
    #validate given number
    if(False == validatePositiveInteger(num)):
        return

    digits = getDigits(num)

    if(type(digits)==int):
        return num

    #create cube of every item of the list using map
    digits = list(map(lambda x:x**3, digits))

    #get cube sum using reduce
    return reduce(lambda x,y : x+y, digits)

def isArmstrong(num):
    """Find whether number is an Armstrong number or not

    Args:
        int
    Returns:
        Boolean True otherwise False
```

```
    """

    if(0 == num or 1 == num):
        return True

    if(num == cubesum(num)):
        return True

    return False


def printArmstrong(minimum=0, maximum=999):
    """Get Armstrong numbers with in a range

    Args:
        minimum: starting number of the list.
        maximum: ending number of the list.
    Returns:
        list
    """

    armstrong = []
    items = list(range(minimum, maximum + 1))
    for i in range(minimum, maximum + 1):
        if(isArmstrong(i)):
            armstrong.append(i)

    return armstrong

print('\n\n')
print('************************* Output of question 6 ****************
**********')
print('\n\n')
# Question 6 driver
# print(isArmstrong(208))
num1 = int(input('Enter a valid number'))
num2 = int(input('Enter a valid number'))
print(printArmstrong(num1, num2))
```

```
************************* Output of question 6 ************************
***



Enter a valid number1
Enter a valid number1000
[1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
```

In [11]:

```python
# 7. Write a function prodDigits() that inputs a number and returns the
 product of digits of that number.

def prodDigits(num):
    """Returns the product of digits of that number.

    Args:
        num: int, given number
    Returns:
        int
    """

    if(False == validatePositiveInteger(num)):
        return

    digits = getDigits(num)

    if(type(digits) == int):
        return num

    multipe = reduce(lambda x, y: x * y, digits)

#     multipe = reduce(multiply, digits)

    return  multipe

print('\n\n')
print('************************* Output of question 7 ***************
**********')
```

```
print('\n\n')
# Question 7 driver
num = int(input('Enter a valid number'))
print(prodDigits(num))
```

```
************************* Output of question 7 ************************
***



Enter a valid number235
30
```

In [12]:

```
# 8. If all digits of a number n are multiplied by each other repeating
 with the product,
# the one digit number obtained at last is called the multiplicative di
gital root of n.
# The number of times digits need to be multiplied to reach one digit i
s called the multiplicative persistance of n.
# Example: 86 -> 48 -> 32 -> 6 (MDR 6, MPersistence 3) 341 -> 12->2 (MD
R 2, MPersistence 2)
# Using the function prodDigits() of previous exercise write functions
 MDR() and MPersistence()
# that input a number and return its multiplicative digital root and mu
ltiplicative persistence respectively

def getMDRAndMPersistence(num):
    """Get multiplicative digital root and multiplicative persistence o
f the given number.

    Args:
        int
    Returns:
        List<MDR, MPersistence>
    """
    cnt=0
    while(True):
```

```
        cnt += 1
        prodOfDigits = prodDigits(num)
        if(prodOfDigits < 10):
            return [prodOfDigits, cnt]
        num = prodOfDigits

    return [prodOfDigits, cnt]

def MDR(num):
    """Get multiplicative digital root of the given number.

    Args:
        int
    Returns:
        int
    """
    if(num < 10):
        return num
    return getMDRAndMPersistence(num)[0]

def MPersistence(num):
    """Get multiplicative persistence of the given number.

    Args:
        int
    Returns:
        int
    """
    if(num <= 10):
        return 1
    return getMDRAndMPersistence(num)[1]

print('\n\n')
print('************************** Output of question 8 ****************
**********')
print('\n\n')
# Question 8 driver
num = int(input('Enter a valid number'))
print(MDR(num))
```

```
num = int(input('Enter a valid number'))
print(MPersistence(num))
```

```
************************* Output of question 8 ************************
***



Enter a valid number86
6
Enter a valid number86
3
```

In [13]:
```
# 9. Write a function sumPdivisors() that finds the sum of proper divis
ors of a number.
# Proper divisors of a number are those numbers by which the number is
 divisible, except the number itself.
# For example proper divisors of 36 are 1, 2, 3, 4, 6, 9, 18

def sumPdivisors(num):
    """Get sum of proper divisors of the given numbers.

    Args:
        num: int, given number
    Returns:
        int
    """
    if(False == validatePositiveInteger(num)):
        return

    divisors = getDivisors(num)
    if(type(divisors) == int):
        return divisors

    return reduce((lambda x, y: x+y), divisors)
```

```
print('\n\n')
print('************************* Output of question 9 ***************
**********')
print('\n\n')
# Question 9 driver
num = int(input('Enter a valid number'))
print(sumPdivisors(num))
```

```
************************* Output of question 9 ***********************
***



Enter a valid number36
55
```

In [14]:
```
# 10. A number is called perfect if the sum of proper divisors of that
 number is equal to the number.
# For example 28 is perfect number, since 1+2+4+7+14=28.
# Write a program to print all the perfect numbers in a given range

def getPerfectNumbers(minimum, maximum):
    """Get perfect numbers within a range.

    Args:
        minimum: starting number of the list.
        maximum: ending number of the list.
    Returns:
        list
    """
    if(False == validatePositiveInteger(minimum)):
        return
    if(False == validatePositiveInteger(maximum)):
        return

    perfectNumbers = []
    for i in range(minimum, maximum):
```

```
        if(i == sumPdivisors(i)):
            perfectNumbers.append(i)

    return perfectNumbers

print('\n\n')
print('************************** Output of question 10 ***************
***********')
print('\n\n')
# Question 10 driver
num1 = int(input('Enter a minimum number'))
num2 = int(input('Enter a maximum number'))
print(getPerfectNumbers(num1, num2))
```

```
************************** Output of question 10 ***********************
****



Enter a minimum number1
Enter a maximum number100
[1, 6, 28]
```

In [15]:
```
# 11. Two different numbers are called amicable numbers
# if the sum of the proper divisors of each is equal to the other numbe
r.
# For example 220 and 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 function to print pairs of amicable numbers in a range

def getAmicableNumber(minimum, maximum):
    """Get amicable numbers b/w given range

    Args:
        minimum: starting number of the list.
        maximum: ending number of the list.
```

```
    Returns:
        list of tuples
    """

    #validate given numbers
    if(False == validatePositiveInteger(minimum)):
        return
    if(False == validatePositiveInteger(maximum)):
        return

    #generate list
    items = range(minimum, maximum + 1)

    amicableNumbers = []
    sumOfProperDivisors = {}

    for i in items:
        #discard prime numbers
        if(True == isPrime(i)):
            continue
        #Sum of proper divisors of given number
        sumPGivenNumber = sumPdivisors(i)
        #Sum of proper divisors of given number
        sumOfSumPGivenNumber = sumPdivisors(sumPGivenNumber)

        if(i == sumOfSumPGivenNumber and i != sumPGivenNumber):
            #List is empty append item without check
            if(0 == len(amicableNumbers)):
                amicableNumbers.append(tuple([i, sumPGivenNumber]))
            else:
                #Check previous index values
                previousElement = amicableNumbers[len(amicableNumbers)-
1]
                if((previousElement[0] + previousElement[1]) != (i + su
mPGivenNumber)):
                    amicableNumbers.append(tuple([i, sumPGivenNumber]))

    return amicableNumbers
```

```
print('\n\n')
print('************************* Output of question 11 ***************
***********')
print('\n\n')
# Question 11 driver
num1 = int(input('Enter a valid number'))
num2 = int(input('Enter a valid number'))
print(getAmicableNumber(num1, num2))
```

```
************************* Output of question 11 ***********************
****



Enter a valid number200
Enter a valid number300
[(220, 284)]
```

In [16]:

```
# 12. Write a program which can filter odd numbers in a list by using f
ilter function
def findOddNumbers(minimum, maximum):
    """Filter odd numbers from the list.

    Args:
        minimum: starting number of the list.
        maximum: ending number of the list.
    Returns:
        list
    """
    if(False == validatePositiveInteger(minimum)):
        return
    if(False == validatePositiveInteger(maximum)):
        return

    numbers = range(minimum, maximum + 1)

    return list(filter(lambda x : (x % 2 != 0), numbers))
```

```
print('\n\n')
print('************************* Output of question 12 ***************
***********')
print('\n\n')
# Question 12 driver
num1 = int(input('Enter a valid number'))
num2 = int(input('Enter a valid number'))
print(findOddNumbers(num1, num2))
```

```
************************* Output of question 12 ***********************
****



Enter a valid number1
Enter a valid number100
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37,
39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73,
75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]
```

In [17]:

```
# 13. Write a program which can map() to make a list whose elements are
 cube of elements in a given list
def cubeListItems(minimum, maximum):
    """Create cube of the list items using map.

    Args:
        minimum: starting number of the list.
        maximum: ending number of the list.
    Returns:
        list
    """
    #validate given numbers
    if(False == validatePositiveInteger(minimum)):
        return
    if(False == validatePositiveInteger(maximum)):
        return
```

```
    #generate list
    items = range(minimum, maximum + 1)

    return list(map(lambda x: x**3, items))

print('\n\n')
print('************************* Output of question 13 ***************
***********')
print('\n\n')
# Question 13 driver
num1 = int(input('Enter a valid number'))
num2 = int(input('Enter a valid number'))
print(cubeListItems(num1, num2))
```

```
************************* Output of question 13 ***********************
****



Enter a valid number1
Enter a valid number100
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744, 3
375, 4096, 4913, 5832, 6859, 8000, 9261, 10648, 12167, 13824, 15625, 17
576, 19683, 21952, 24389, 27000, 29791, 32768, 35937, 39304, 42875, 466
56, 50653, 54872, 59319, 64000, 68921, 74088, 79507, 85184, 91125, 9733
6, 103823, 110592, 117649, 125000, 132651, 140608, 148877, 157464, 1663
75, 175616, 185193, 195112, 205379, 216000, 226981, 238328, 250047, 262
144, 274625, 287496, 300763, 314432, 328509, 343000, 357911, 373248, 38
9017, 405224, 421875, 438976, 456533, 474552, 493039, 512000, 531441, 5
51368, 571787, 592704, 614125, 636056, 658503, 681472, 704969, 729000,
753571, 778688, 804357, 830584, 857375, 884736, 912673, 941192, 970299,
1000000]
```

In [18]:
```
# 14. Write a program which can map() and filter() to
# make a list whose elements are cube of even number in a given list
def cubeEvenItemsOfList(minimum, maximum):
```

```
    """Use map and filter to generate cube of each even items of the li
st.

    Args:
        minimum: starting number of the list.
        maximum: ending number of the list.
    Returns:
        list
    """
    #validate given numbers
    if(False == validatePositiveInteger(minimum)):
        return
    if(False == validatePositiveInteger(maximum)):
        return

    #generate list
    items = range(minimum, maximum + 1)

    #first filter even items then create cube of those
    return list(map(lambda x: x**3, list(filter(lambda x: (x % 2 == 0),
 items))))

print('\n\n')
print('************************** Output of question 14 ***************
***********')
print('\n\n')
# Question 14 driver
num1 = int(input('Enter a valid number'))
num2 = int(input('Enter a valid number'))
print(cubeEvenItemsOfList(num1, num2))
```

```
************************** Output of question 14 ***********************
****



Enter a valid number1
Enter a valid number100
```

```
[8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832, 8000, 10648, 13824, 175
76, 21952, 27000, 32768, 39304, 46656, 54872, 64000, 74088, 85184, 9733
6, 110592, 125000, 140608, 157464, 175616, 195112, 216000, 238328, 2621
44, 287496, 314432, 343000, 373248, 405224, 438976, 474552, 512000, 551
368, 592704, 636056, 681472, 729000, 778688, 830584, 884736, 941192, 10
00000]
```