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

In [16]:

```
# get the input from the user, Support only Numeric value

def create_multiplication_table():
    try:
        get_input =  int(input('Please enter the Numeric input, for multliplication table,

        if get_input == 0:
            print('Thank you :)')
            return

        for i in range(1,11):
            print(i,'X',get_input,'  = ',i*get_input)
        create_multiplication_table()

    except:
        print('Sorry!, please enter Numeric value')
        create_multiplication_table()


create_multiplication_table()
```

...

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

In [48]:

```
'''
Two prime numbers are called twin primes if there is present only one composite number betw
Or we can also say two prime numbers whose difference is two are called twin primes. For ex
since the difference between the two numbers 5 – 3 = 2.
The alternative names, given to twin primes are prime twin or prime pair. Also, learn prime

'''

# First find list of prime number
def prime(inputs):
    prime_number=list(range(2,inputs))
    for i in range(2,inputs+1):
        for j in range(2,inputs+1):
            if i != j:
                if i%j == 0:
                    try:
                        prime_number.remove(i)
                    except:
                        pass
                    break
    return prime_number


# Find twin prime
def twin_prime(prime):
    twin_prime_val =[]
    for i in prime:
        for j in prime:
            if j-i == 2:
                twin_prime_val.append((i,j))
    return twin_prime_val


prime_numbers = prime(int(input('Enter your range, to find twin prime number')))
twin_prime(prime_numbers)
```

```
Enter your range, to find twin prime number1000
```

Out[48]:

```
[(3, 5),
 (5, 7),
 (11, 13),
 (17, 19),
 (29, 31),
 (41, 43),
 (59, 61),
 (71, 73),
 (101, 103),
 (107, 109),
 (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),
(569, 571),
(599, 601),
(617, 619),
(641, 643),
(659, 661),
(809, 811),
(821, 823),
(827, 829),
(857, 859),
(881, 883)]
```

3. Write a program to find out the prime factors of a number. Example: prime factors of 56 -2,2, 2, 7

Following are the steps to find all prime factors.
1) While n is divisible by 2, print 2 and divide n by 2.
2) After step 1, n must be odd. Now start a loop from i = 3 to square root of n. While i divides n, print i and divide n by i. After i fails to divide n, increment i by 2 and continue.
3) If n is a prime number and is greater than 2, then n will not become 1 by above two steps. So print n if it is greater than 2.

In [20]:

```
import numpy as np
def prime_or_not(inputs,display=False):
    prime_yes =True
    for j in range(2,inputs+1):
        if inputs != j:
            if inputs % j == 0:
                if display:
                    print('Given number if Not a prime Number, because is divisible of {}.
                prime_yes = False
                break
    if prime_yes == True:
        if display:
            print('Given Number {} is Prime Number'.format(inputs))
    return prime_yes

def step1_2(n):
    if prime_or_not(n) == False:
        if n%2 ==0: # Step 1
            print(2)
            n = int(n/2)
            step1_2(n)
        else: # Step 2
            for k in range(3, int(np.sqrt(n))+1,2):
                if n%k ==0:
                    print(k)
                    n = int(n/k)
                    step1_2(n)
                    break


    else:
        print(n)
        if prime_or_not(n) == True:
            print('Note {} is Prime Number, so we cant further divide.'.format(n))



print('Prime factor of 56',)
step1_2(56)
print('**'*50)
print('Prime factor of 315',)
step1_2(315)
print('**'*50)
print('Prime factor of 34866',)
step1_2(34866)
```

```
Prime factor of 56
2
2
2
7
Note 7 is Prime Number, so we cant further divide.
****************************************************************************************************
Prime factor of 315
3
3
```

```
5
7
Note 7 is Prime Number, so we cant further divide.
************************************************************************
************************
Prime factor of 34866
2
3
3
13
149
Note 149 is Prime Number, so we cant further divide.
```

4. Write a program to implement these formulae of permutations and 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 is: c(n, r) = n! / (r!*(n-r)!) = p(n, r) / r!

In [83]:

```
def permutation(list_num):
    #print(list_num)

    # If list_num is empty then there are no permutations
    if len(list_num) ==0:
        return []
    #Only one permutation is possible
    if len(list_num) == 1:
        return [list_num]

    #if list is greater than 1
    new_list = []

    for x in range(len(list_num)):
        m = list_num[x]
        remaining_list = list_num[:x] +list_num[x+1:]
        #print(remaining_list)

        # Generating all permutations where m is first element
        for p in permutation(remaining_list):
            new_list.append([m] + p)
    return   new_list


 # Permutation of given number
data = list('123')
for p in permutation(data):
    print(p)
```

```
['1', '2', '3']
['1', '3', '2']
['2', '1', '3']
['2', '3', '1']
['3', '1', '2']
['3', '2', '1']
```

In [84]:

```
def combination(lst,n):
    if n==0:
        return [[]]
    l=[]
    for i in range(0,len(lst)):
        m=lst[i]
        remLst=lst[i+1:]
        for p in combination(remLst,n-1):
            l.append([m]+p)
    return l

list(combo2([1,2,3,4,5],3))
```

Out[84]:

```
[[1, 2, 3],
 [1, 2, 4],
 [1, 2, 5],
 [1, 3, 4],
 [1, 3, 5],
 [1, 4, 5],
 [2, 3, 4],
 [2, 3, 5],
 [2, 4, 5],
 [3, 4, 5]]
```

5. Write a function that converts a decimal number to bina ry number

In [26]:

```
def convert_Dec_to_Binary(n):
    if n >1:
        convert_Dec_to_Binary(n//2) # // - return Integer
    print(n%2,end='')

convert_Dec_to_Binary(101)
```

```
1100101
```

**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 PrintArmstro ng() and
isArmstrong() to print Armstrong numbers and to find whether is an Armstrong numbe r.**

In [15]:

```
def cubesum(x):
    # initialize sum
    sum = 0

    # find the sum of the cube of individual digit
    temp = x
    while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
    return sum

def printArmstrong(x):
    cube_sum = cubesum(x)
    if cube_sum == x:
        print(x, 'is Armstrong number')
    else:
        print(x, 'is not armstrong number')


printArmstrong(850)
printArmstrong(153)
```

```
850 is not armstrong number
153 is Armstrong number
```

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

In [24]:

```
def get_prod(x):
    prod = 1

    temp = x
    while temp !=0:
        prod = prod*(temp%10)
        temp = temp//10
    print('The product of digits {} is {}'.format(x,prod))


get_prod(153)
get_prod(8496)
```

```
The product of digits 153 is 15
The product of digits 8496 is 1728
```

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 digital root of n. The

**number of
times digits need to be multiplied to reach one digit is called the multiplicative persistance of n.
Example: 86 -> 48 -> 32 -> 6 (MDR 6, MPersistence 3)
341 -> 12->2 (MDR 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 multiplicative persistence respectively**

In [51]:

```
def get_prod(x):
    prod = 1

    temp = x
    while temp !=0:
        prod = prod*(temp%10)
        temp = temp//10
    return prod

def MDR(x):
    count = 0
    length = 0
    temp =x
    while length !=1:
        temp = get_prod(temp)
        length = len(str(temp))
        count+=1
    print('multiplicative digital root of {} is {}'.format(x,temp))
    print('multiplicative persistence of {} is {}'.format(x,count))


#Starting
MDR(86)
print('**'*50)
MDR(341)
```

```
multiplicative digital root of 86 is 6
multiplicative persistence of 86 is 3
**********************************************************************
**********
multiplicative digital root of 341 is 2
multiplicative persistence of 341 is 2
```

**9. Write a function sumPdivisors() that finds the sum of proper divisors 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, 9,12, 18**

In [18]:

```
def sumPdivisor(num):
    divisor_list =[]
    for i in range(1,num):
        if num%i == 0:
            divisor_list.append(i)
    print('The sum of proper divisors of a number are ', divisor_list)

num= int(input('Enter the number to find sum of proper divisor - '))
sumPdivisor(num)
```

```
Enter the number to find sum of proper divisor - 28
The sum of proper divisors of a number are  [1, 2, 4, 7, 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

In [26]:

```
def Prefect_number(num):
    divisor_list =[]
    for i in range(1,num):
        if num%i == 0:
            divisor_list.append(i)
    print('The sum of proper divisors of a number are ', divisor_list)
    if sum(divisor_list) == num:
        print('The number {} is perfect number'.format(num))
    else:
        print('The number {} is Not a perfect number, because sum of proper divisor is {}'.

num= int(input('Enter the number to find proper number or not?  - '))
Prefect_number(num)
```

```
Enter the number to find proper number or not?  - 28
The sum of proper divisors of a number are  [1, 2, 4, 7, 14]
The number 28 is perfect number
```

11. Two different numbers are called amicable numbers if the sum of the proper divisors of each is equal to the other number. 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

In [33]:

```
def sumPdivisor(num):
    divisor_list =[]
    for i in range(1,num):
        if num%i == 0:
            divisor_list.append(i)
    return  divisor_list



def amicable_number(num):
    list_divisor = sumPdivisor(num)
    sum_proper_divisor =sum(list_divisor)
    if sum_proper_divisor != num:
        list_divisor_new = sumPdivisor(sum_proper_divisor)
        sum_proper_divisor_new =sum(list_divisor_new)
        if sum_proper_divisor_new == num:
            print('Numbers {} and {} are amicable numbers'.format(num,sum_proper_divisor))
        else:
            print('Numbers {} and {} are not amicable numbers'.format(num,sum_proper_diviso




num= int(input('Enter the number to find amicable numbers or not?  - '))
amicable_number(num)
```

```
Enter the number to find amicable numbers or not?  - 220
Numbers 220 and 284 are amicable numbers
```

12. Write a program which can filter odd numbers in a list by using filter functi on

In [37]:

```
# Initialisation of list
lis1 = range(1,31)

is_odd = lambda x: x % 2 != 0

# using filter function
lis2 = list(filter(is_even, lis1))

# Printing output
print(lis2)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
```

13. Write a program which can map() to make a list whose elements are cube of elem ents in a given list

In [45]:

```
def find_cube(num):
    '''
    Cube function created
    '''
    return num**3

list_of_numbers= range(1,10) # given list of numbers
map_function = map(find_cube,list_of_numbers) #  Cube function is mapped
print(list(map_function))
```

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

**14. Write a program which can map() and filter() to make a list whose elements are cube of
even number in a given list**

In [46]:

```
def find_cube(num):
    '''
    Cube function created
    '''
    return num**3

list_of_numbers= range(1,10) # given list of numbers
map_function = map(find_cube,list_of_numbers) #  Cube function is mapped
cube_list = list(map_function)

is_even = lambda x: x % 2 == 0
# using filter function
lis2 = list(filter(is_even, cube_list))

# Printing output
print(lis2)
```

```
[8, 64, 216, 512]
```

################################## XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ############## ############################