In [3]:
```
#1. Write a function that inputs a number and prints the multiplication table of that number
def mult(num):                                  #Function implementation
    for multiplier in range(1,11):
        print('{0} * {1} = {2}'.format(num,multiplier,num*multiplier))

mult(int(input('Enter the Number to fetch its Table :')))  #Function Call
```

```
Enter the Number to fetch its Table :7
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

In [11]:
```
#2. Write a program to print twin primes less than 1000.

prime_num=[]
twin_prime=[]
for num in range(3,1001):      # As 2 is neither prime nor composite, so started from 3
    #pdb.set_trace()
    for i in range(2,num):
        if(num%i==0):
            break
    else:
        prime_num.append(num)
        if len(prime_num)>1:
            twin_prime.append((prime_num[-2],prime_num[-1]))
print('List of Twin Primes:')
print(twin_prime)
```

```
List of Twin Primes:
[(3, 5), (5, 7), (7, 11), (11, 13), (13, 17), (17, 19), (19, 23), (23, 29), (29, 31), (31, 37),
(37, 41), (41, 43), (43, 47), (47, 53), (53, 59), (59, 61), (61, 67), (67, 71), (71, 73), (73, 7
9), (79, 83), (83, 89), (89, 97), (97, 101), (101, 103), (103, 107), (107, 109), (109, 113), (11
3, 127), (127, 131), (131, 137), (137, 139), (139, 149), (149, 151), (151, 157), (157, 163), (16
3, 167), (167, 173), (173, 179), (179, 181), (181, 191), (191, 193), (193, 197), (197, 199), (19
9, 211), (211, 223), (223, 227), (227, 229), (229, 233), (233, 239), (239, 241), (241, 251), (25
1, 257), (257, 263), (263, 269), (269, 271), (271, 277), (277, 281), (281, 283), (283, 293), (29
3, 307), (307, 311), (311, 313), (313, 317), (317, 331), (331, 337), (337, 347), (347, 349), (34
9, 353), (353, 359), (359, 367), (367, 373), (373, 379), (379, 383), (383, 389), (389, 397), (39
7, 401), (401, 409), (409, 419), (419, 421), (421, 431), (431, 433), (433, 439), (439, 443), (44
3, 449), (449, 457), (457, 461), (461, 463), (463, 467), (467, 479), (479, 487), (487, 491), (49
1, 499), (499, 503), (503, 509), (509, 521), (521, 523), (523, 541), (541, 547), (547, 557), (55
7, 563), (563, 569), (569, 571), (571, 577), (577, 587), (587, 593), (593, 599), (599, 601), (60
1, 607), (607, 613), (613, 617), (617, 619), (619, 631), (631, 641), (641, 643), (643, 647), (64
7, 653), (653, 659), (659, 661), (661, 673), (673, 677), (677, 683), (683, 691), (691, 701), (70
1, 709), (709, 719), (719, 727), (727, 733), (733, 739), (739, 743), (743, 751), (751, 757), (75
7, 761), (761, 769), (769, 773), (773, 787), (787, 797), (797, 809), (809, 811), (811, 821), (82
1, 823), (823, 827), (827, 829), (829, 839), (839, 853), (853, 857), (857, 859), (859, 863), (86
3, 877), (877, 881), (881, 883), (883, 887), (887, 907), (907, 911), (911, 919), (919, 929), (92
9, 937), (937, 941), (941, 947), (947, 953), (953, 967), (967, 971), (971, 977), (977, 983), (98
3, 991), (991, 997)]
```

In [9]:
```
#3. Write a program to find out the prime factors of a number
num = int(input('Enter the number to find its factor : '))
print_num=num;
list1=[]

while num!=1:   #Restart Logic implemented to restart for loop
    restart=False
    for i in range(2,num+1):
        if(num%i==0):
            num = num//i
            list1.append(i)
            restart=True   #once the the 1st factor is found, force the code to find the second fact
or from 2 onwards
            break
print("Factors of {0} = {1}".format(print_num,list1))
```

```
Enter the number to find its factor : 162
Factors of 162 = [2, 3, 3, 3, 3]
```

In [ ]:
```
#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!

n=int(input('Enter the number of elements : '))
r=int(input('Enter the samples : '))

def factorial(num):
    return 1 if num==1 else num*factorial(num-1)

permutation=factorial(n)/factorial(n-r)
combination=permutation/factorial(r)
print('Probability and Combination of P({0},{1}) is {2} and {3}'.format(n,r,permutation,combination
))
```

In [14]:
```
#5. Write a function that converts a decimal number to binary number
list1=[]
def dec2bin(num):
    list1.append(num%2)
    num=num//2
    if num==1:
        list1.append(1)
    return list1 if num==1 else dec2bin(num)

a=dec2bin(int(input('enter the Decimal number : ')))
a.reverse()
print(''.join(map(str,a)))
```

```
enter the Decimal number : 20
10100
```

In [17]:
```
#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 number.
def cubesum(num,b=0):
    a=divmod(num,10)
    num=a[0]
    b=b+a[1]**3
    return b if num==0 else cubesum(num,b)

def isArmstrong(calculated_cube,num):
    return True if calculated_cube==num else False

def PrintArmstrong(dec,num):
    print('Number {0} is an Armstrong Number'.format(num)) if dec else print('Number {0} is Not an A
rmstrong Number'.format(num))

num=int(input('Enter no to verify Armstong : '))
PrintArmstrong(isArmstrong(cubesum(num),num),num)
```

```
Enter no to verify Armstong : 160
Number 160 is Not an Armstrong Number
```

In [16]:
```
#7. Write a function prodDigits() that inputs a number and returns the product of digits of that num
ber.
from functools import reduce
def prodDigits(str_num):
    return reduce(lambda x,y:x*y,[int(ele) for ele in str_num])   #Convert string into list of integ
er and then applied the Lambda function

a=input('Enter  the number : ')
print('Product of {0} is {1}'.format(int(a),prodDigits(a)))
```

```
Enter  the number : 4567
Product of 4567 is 840
```

In [18]:
```
#8. 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
#Example: 86 -> 48 -> 32 -> 6 (MDR 6, MPersistence 3)

from functools import reduce
def prodDigits(str_num):
    return reduce(lambda x,y:x*y,[int(ele) for ele in str_num])

def MDR(a,count=0):
    count=MPersistence(count)
    a=prodDigits(str(a))
    return (a,count) if len(str(a))==1 else MDR(a,count)
    #return a,count if len(str(a))==1 else MDR(a,count)  #Doubt p If i try to runt his code , them t
he op is tuples of tuples.

def MPersistence(count):
    count=count+1
    return count

a=b=input('Enter the number : ')
op=MDR(a)
print('MDR and MPersistence of {0} is {1} and {2} respectively'.format(b,op[0],op[1]))
```

```
Enter the number : 123456
MDR and MPersistence of 123456 is 0 and 2 respectively
```

In [9]:
```
#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
from functools import reduce
def sumPdivisors(num):
    return reduce(lambda x,y:x+y,[ele for ele in range(1,(num//2)+1) if num%ele==0])

a=int(input('Enter the Number :'))
print('Sum of proper divisor of number {0} is {1}'.format(a,sumPdivisors(a)))
```

```
Enter the Number :36
Sum of proper divisor of number 36 is 55
```

In [1]:
```
#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
from functools import reduce
low=int(input('Enter the Low Number :'))
high=int(input('Enter the High Number :'))


lst=[]
for range_ele in range(low,high+1):
    a=0
    for ele in range(1,(range_ele//2) +1):
        #pdb.set_trace()
        if range_ele%ele==0:
            a=a+ele
    if a==range_ele:
        lst.append(a)
print('Perfect number between {0} and {1} are {2}'.format(low,high,lst))
```

```
Enter the Low Number :1
Enter the High Number :10000
Perfect number between 1 and 10000 are [6, 28, 496, 8128]
```

In [3]:
```
#10  solution 2. 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 sumPdivisors(num):
    return reduce(lambda x,y:x+y,[ele for ele in range(1,(num//2)+1) if num%ele==0])

def perfect_number(num):            #This Returns the a number if its perfect
    return num if (sumPdivisors(num)==num) else None

from functools import reduce
low=int(input('Enter the Low Number :'))
high=int(input('Enter the High Number :'))

'''"if i>1" is used because error 'reduce() of empty sequence with no initial value'
 is thrown -> in the lambda function only x is present and not y for the initial iteration'''
number=[i for i in range(low,high+1) if i>1]
a=list(filter(perfect_number,number))
print('Perfect number between {0} and {1} are {2}'.format(low,high,a))
```

```
Enter the Low Number :1
Enter the High Number :1000
Perfect number between 1 and 1000 are [6, 28, 496]
```

In [11]:
```
#11.Write a function to print pairs of amicable numbers in a range
#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
def sumOfDiv(x):
    sum = 1
    for i in range(2, x):
        if x % i == 0:
            sum += i
    return sum

def isAmicable(a, b):
    if sumOfDiv(a) == b and sumOfDiv(b) == a:
        return True
    else:
        return False

def countPairs(arr, n):
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            if isAmicable(arr[i], arr[j]):
                count = count + 1
                amicable_list.append((arr[i], arr[j]))
    return count


a = int(input('Enter Lower range : '))
b = int(input('Enter upper range : '))
amicable_list=[]
# Driver Code
arr = [num for num in range(a,b+1)]
arr_len = len(arr)
print('Count of Amicable number in range {0} and {1} is {2} and the pairs of amicable numbers is {3}
'.format(a,b,countPairs(arr, arr_len),amicable_list))
```

```
Enter Lower range : 10
Enter upper range : 3000
Count of Amicable number in range 10 and 3000 is 3 and the pairs of amicable numbers is [(220, 2
84), (1184, 1210), (2620, 2924)]
```

In [5]:
```
#12. Write a program which can filter odd numbers in a list by using filter function
num_list=[i for i in range(0,100)]
odd_list=list(filter(lambda x:(x%2!=0),num_list))
print('Odd List\n',odd_list)
```

```
Odd List:
 [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 [6]:
```
#13.Write a program which can map() to make a list whose elements are cube of elements in a given li
st
num_list=[i for i in range(0,100)]
cube_list=list(map(lambda x:x**3,num_list))
print(cube_list)
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744, 3375, 4096, 4913, 5832,
6859, 8000, 9261, 10648, 12167, 13824, 15625, 17576, 19683, 21952, 24389, 27000, 29791, 32768, 3
5937, 39304, 42875, 46656, 50653, 54872, 59319, 64000, 68921, 74088, 79507, 85184, 91125, 97336,
103823, 110592, 117649, 125000, 132651, 140608, 148877, 157464, 166375, 175616, 185193, 195112,
205379, 216000, 226981, 238328, 250047, 262144, 274625, 287496, 300763, 314432, 328509, 343000,
357911, 373248, 389017, 405224, 421875, 438976, 456533, 474552, 493039, 512000, 531441, 551368,
```