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

In [1]:
```
def mult_table():
    n= int(input("enter number: "))
    m= int(input("multiplication table till? "))
    for i in range(1,m+1):
        p=n * i
        print('{} * {} = {}'.format(n,i,p))


mult_table()
```

```
enter number: 5
multiplication table till? 10
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

1. 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 [2]:
```
import math

answer=[]

for i in range(1,1000,2):
    #store 2 consecutive odd numbers in a and b
    a= i
    b= i+2
    c= 2
    flag = 1
    #check divisibility till square root of b
    while c < math.sqrt(b+1):
        if a % c == 0 or b % c == 0:
            flag = 0
            break
        else: c+=1
    # if both a and b are prime, add it to list
    if flag == 1: answer.append((a,b))

print(answer)
```

```
[(1, 3), (3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (7
1, 73), (101, 103), (107, 109), (137, 139), (149, 151), (179, 181), (191, 19
3), (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)]
```

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

In [4]:
```
import math

def is_prime(num):
    '''function to check if number is prime'''
    flag =1
    for i in range(2,int(math.sqrt(num)+1)):
        if num%i==0: flag =0
    return flag

n=int(input("enter number: "))

factor_list=[]
#list of prime factors of n to be generated
i=2
while i <n+1 :
    flag= is_prime(i)

    if flag == 1:
        if n%i==0:
            #i is prime and divisible by n, then add to prime factors list
            factor_list.append(i)
            n=n/i
            #n replaced by the next smallest composite factor and process cont
iinued
            i=2
        else: i+=1
    else: i+=1



print(factor_list)
```

```
enter number: 327
[3, 109]
```

1. 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 [5]:
```
n= int(input("enter n: "))
r= int(input("enter r:" ))

def fact(num):
    '''function to compute factorial of number'''
    if num ==1 or num ==0:
        factorial= 1
    else: factorial= num*fact(num-1)
    return factorial

perm = fact(n)/fact(n-r)
comb = perm/ fact(r)

print('permitations: {}, combinations: {}'.format(perm,comb))
```

```
enter n: 5
enter r:3
permitations: 60.0, combinations: 10.0
```

1. Write a function that converts a decimal number to binary number

In [16]:
```
binary=""

n= int(input("decimal number: "))

q=n
r=0

while q>0:
    #calculating floor(quotient) and remainder on division by 2
    r= q%2
    q= q//2
    #remainder added to binary string
    binary+=str(r)

#print the string of remainders
print('binary conversion: {}'.format(binary[::-1]))
```

```
decimal number: 10
binary conversion: 1010
```

1. 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.

In [25]:
```
def cubesum (num):
    n = str(num)
    s =0
    for i in range(len(n)):
        s= s + pow(int(n[i]),3)
    return s



#print('cubesum = {}'.format(s))

def is_Armstrong(num):
    s = cubesum(num)
    if s==num:
        return 1
    else: return 0


#print(is_Armstrong(num))

def print_Armstrong(n):
    armstrong_numbers = []
    for i in range (n):
        if is_Armstrong(i) ==1:
            armstrong_numbers.append(i)
    print('armstrong numbers less than {} are {}'.format(n,armstrong_numbers))

n = int(input("armstrong numbers less than: "))
print_Armstrong(n)
```

```
armstrong numbers less than: 1000
armstrong numbers less than 1000 are [0, 1, 153, 370, 371, 407]
```

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

In [27]:
```
num = int(input("enter number: "))

num_str = str(num)
product = 1
for i in range(len(num_str)):
    product = product * int(num_str[i])

print('product of digits of number is {}'.format(product))
```

```
enter number: 1023
product of digits of number is 0
```

1. 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 [29]:
```
num = int(input("enter number: "))

def prod_digits(num):
    num_str = str(num)
    product = 1
    for i in range(len(num_str)):
        product = product * int(num_str[i])
    return (product)

def MDR_MPersistence(num):
    i =0
    while num >9:
        num = prod_digits(num)
        i += 1
    return (num, i)

MDR, MPersistence = MDR_MPersistence(num)

print ('MDR: {}, MPersistence: {}'.format(MDR,MPersistence))
```

```
enter number: 86
MDR: 6, MPersistence: 3
```

1. 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 [31]:
```
num= int(input("enter number: "))
sum = 0

for i in range(1,num):
    if (num%i==0):
        #print(i)
        sum +=i

print('sum of proper divisors of {} is {}'.format(num, sum))
```

```
enter number: 36
1
2
3
4
6
9
12
18
sum of proper divisors of 36 is 55
```

1. 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 [33]:
```
n= int(input('range start: '))
m= int(input('range end: '))

def factor_sum(num):
    sum = 0
    for i in range(1,num):
        if (num%i==0): sum +=i
    return sum

perfect_number =[]
for i in range(n,m+1):
    if i == factor_sum(i):
        perfect_number.append(i)

print('Perfect numbers in range {} to {} is {}'. format(n,m,perfect_number))
```

```
range start: 1
range end: 50
Perfect numbers in range 1 to 50 is [6, 28]
```

1. 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 [37]:
```
def factor_sum(num):
    sum = 0
    for i in range(1,num):
        if (num%i==0): sum +=i
    return sum

n= int(input('range start: '))
m= int(input('range end: '))

factor_sums =[]

for i in range(n,m+1):
    factor_sums.append(factor_sum(i))

amicable_pairs = []

for i in range(n, m+1):
    j=i+1
    while j< factor_sum(i)+1:
        if i==factor_sum(j) and j== factor_sum(i) and i!=j :
            amicable_pairs.append((i,j))
        j+=1

if len (amicable_pairs)==0:
    print('No amicable pairs in this range')
else:
    print('Amicable Pairs in range {} to {} are {}'.format(n,m, amicable_pairs
))
```

```
range start: 1
range end: 1000
Amicable Pairs in range 1 to 1000 are [(220, 284)]
```

1. Write a program which can filter odd numbers in a list by using filter function

In [40]:
```
array= []
n= int(input("length of list: "))

print('enter values: ')
for i in range(n):
    array.append(int(input()))

def filter_odd(num):
    if num%2== 0:
        return False
    else:
        return True



filtered_array= list(filter(filter_odd, array))

print(filtered_array)
```

```
length of list: 5
enter values:

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

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

In [41]:
```
num = int(input('length of list: '))
array=[]
for i in range(num):
    m = int (input())
    array.append(m)

result= map(lambda x: x*x*x, array)
print(list(result))
```

```
length of list: 5
1
2
3
4
5
[1, 8, 27, 64, 125]
```

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

In [42]:
```
num = int(input('length of list: '))
array=[]
for i in range(num):
    m = int (input())
    array.append(m)


def filter_even(num):
    if num%2== 0:
        return True
    else:
        return False


filtered_array= list(filter(filter_even, array))


result= map(lambda x: x*x*x, filtered_array)
print(list(result))
```

```
length of list: 5
1
2
3
4
5
[8, 64]
```