01 - Write a function that inputs a number and prints the multiplication table of that number

In [1]:
```
def multiplication():
    number = int(input("Enter a number to know its tables:"))
    for items in range(1,11):
        print(number,"*",items,"=", number*items)

multiplication()
```

```
Enter a number to know its tables:4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
```

02 - 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]:
```
def prime_number(number):
    for item in range(2,number):
        if number%item==0:
            return False
    return True

def twin_primes(start,end):
    for item in range(start,end):
        prime_no = item+2
        if(prime_number(item) and prime_number(prime_no)):
            print(f"{item} and {prime_no}")


twin_primes(2,1000)
```

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

03 - Write a program to find out the prime factors of a number.

In [3]:
```
#Example: prime factors of 56 -2, 2, 2, 7

n = int(input("Enter any Number: "))
i = 2
while (n != 1):
    rem = n % i
    if (rem == 0):
        n = n/i
        print(i,end=",")
    else:
        i = i+1
```

```
Enter any Number: 56
2,2,2,7,
```

04 - 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!

Method-1

In [4]:
```
def factorial(n):
    if n < 0:
        return 0
    elif n == 0 or n == 1:
        return 1
    else:
        fact = 1
        while (n > 1):
            fact *= n
            n -= 1
        return fact


def for_values():
    n = int(input("Value for n:"))
    r = int(input("Value for r:"))

    if n < r:
        print("INVALID Entry Value of n should be greater than r")
    else:
        #permutation
        npr = factorial(n) / factorial(n - r)
        print("Permutation of", n, "&", r, "is", int(npr))
        #combination
        comb = npr / factorial(r)
        print("combination of", n, "&", r, "is", int(comb))


for_values()
```

```
Value for n:7
Value for r:4
Permutation of 7 & 4 is 840
Combination of 7 & 4 is 35
```

04 - 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!

Method-2

Using inbuilt Model

In [5]:
```
import math

def for_values():


    n = int(input("Value for n:"))
    r = int(input("Value for r:"))

    if n<r:
        print("INVALID Entry Value of n should be greater than r")
        for_values()
    else:
        #permutation
        npr = math.factorial(n)/math.factorial(n-r)
        print("npr is", int(npr))
        #combination
        ncr = npr/math.factorial(r)
        print("ncr is", int(ncr))

for_values()
```

```
Value for n:7
Value for r:4
npr is 840
ncr is 35
```

05 - Write a function that converts a decimal number to binary number.

In [6]:
```
def decimalToBinary(num):

    if num > 1:
        decimalToBinary(int(num / 2))
    print(num % 2, end='')


# decimal number
number = int(input("Enter any decimal number: "))

decimalToBinary(number)
```

```
Enter any decimal number: 54
110110
```

06 - write a function cubesum() that accepts an integer and returns the sum of the cubes ofindividual digits of that number. Use this function to make functions PrintArmstrong() andisArmstrong() to print Armstrong numbers and to find whether is an Armstrong number.

In [7]:
```
def printarmstrong(number):
    print("Entered value", number, "is a armstrong number")


def isarmstrong(result, number):
    if result == number:
        printarmstrong(number)
    else:
        print("Entered value", number, "is not a armstrong number")


def cubesum():
    number = int(input("Enter a value to find whether it is armstrong number or not:"))
    power = len(str(number))

    if power != 3:
        print("Enter a value which have 3 digits")
        cubesum()
    else:
        rslt = 0
        i = number

        while i != 0:
            reminder = int(i % 10)
            rslt = rslt + (reminder ** 3)
            i = i // 10

    print("Sum of cube of individual digits is ", rslt)
    isarmstrong(rslt, number)



cubesum()
```

```
Enter a value to find whether it is armstrong number or not:543
Sum of cube of individual digits is  216
Entered value 543 is not a armstrong number
```

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

In [8]:
```
def prodDigits(n):
    prod = 1
    temp = n
    while temp > 0:
        prod = prod * (number%10)
        number = int(number/10)

    print("Product of digits of number", n , "is", prod)

prodDigits(56)
```

```
Product of digits of number 56 is 30
```

08 - 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 [9]:
```
def prodDigit(num):
    temp = num
    prod = 1
    while temp > 0:
        digit = temp % 10
        prod *= digit
        temp //= 10
    return prod

def MDR(num):
    digit = num
    pers = 0
    while digit > 9:
        digit =prodDigit(int(digit))
        pers += 1
    return int(digit), pers

num = int(input("Enter a value to know its multiplicative digital root  and multiplicative persistence: "))
digital_root, mper = MDR(num)
print("For {0} Multiplicative digital root is {1} and M Persistance is {2}".format(num, digital_root, mper))
```

```
Enter a value to know its multiplicative digital root  and multiplicative persistence: 654
For 654 Multiplicative digital root is 0 and M Persistance is 2
```

09 - 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, 18

In [10]:
```
def sumPdivisors():
    num = int(input("Enter a value:"))
    sum = 0

    for i in range (1,num):
        if num % i == 0:
            sum = sum +i
    print("Sum of proper divisors of",num , "is", sum)


sumPdivisors()
```

```
Enter a value:65
Sum of proper divisors of 65 is 19
```

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 [11]:
```
def sumPdivisors(num):
    sum = 0
    for i in range (1,num):
        if num % i == 0:
            sum = sum +i
    return sum

def perfect_number(start,end):
    for i in range(start,end):
        if i == sumPdivisors(i):
            print(i)


perfect_number(1,50)
```

```
6
28
```

11 - 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 [12]:
```
def sumPdivisors(num):
    sum = 0
    for i in range (1,num):
        if num % i == 0:
            sum = sum +i
    return sum


def amicable_numbers(start,end):

    for number in range(start,end):
        for ele in range(1,number):
            if number != ele:
                if sumPdivisors(number) == ele and sumPdivisors(ele) == number:
                    print("Amicable numbers in the range", (start,end),"are",number,ele)


amicable_numbers(1, 1000)
```

```
Amicable numbers in the range (1, 1000) are 284 220
```

12 - Write a program which can filter odd numbers in a list by using filter function

In [13]:
```
def odd_list(lst):

    odd_list = list(filter(lambda num: (num%2 != 0),lst))
    print(odd_list)


odd_list(*range(1,100))
```

```
[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]
```

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

In [14]:
```
def cube_list(lst):

    cube_list = list(map(lambda num: num**3,lst))
    print(cube_list)


cube_list(*range(1,10))
```

```
[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 [15]:
```
def cube_list(lst):
    cube_list= list(map(lambda x: x**3, lst))
    print(cube_lst)


def even_list(lst):
    return cube_list(list(filter(lambda num: (num%2) == 0, lst)))


even_list(*range(1,11))
```

```
[8, 64, 216, 512, 1000]
```