PYTHON ASSIGNMENT (optional)

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

In [1]:
```
def multiplication_table():
    """
    This function prints the multiplication table of the given numb
er
    """
    num = int(input("Enter the table number: "))
    limit =int(input("Enter the range for the table: "))
    print("Multiplication table of {} is :".format(num))

    for i in range(1,limit+1):
        print(i,"x", num, "=", num * i)

multiplication_table()
```

```
Enter the table number: 9
Enter the range for the table: 4
Multiplication table of 9 is :
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
```

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 [2]:
```
# to check a number is prime or not
def isPrime(num):
    if num>1:
        for i in range(2,num):
            if num%i == 0:
                break
        else:
            return True

# to print twin primes(pairs of primes which differ by two)less tha
n 1000
print("Twin primes less than 1000 are :")
for i in range(1,1000):
    j = i+2
    if isPrime(i) and isPrime(j):
        print("({},{})".format(i, j))
```

```
Twin primes less than 1000 are :
(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

In [3]:
```
def prime_factors(num):
    """
    This function returns the prime factors of the given number
    """
    factors = [] #list to store prime factors
    i = 2 # lowest prime number

    while i <= num:
        if num%i == 0:
            factors.append(i)
            num = num/i
        else :
            i += 1
    return factors

num =150
print("Prime factor of {} is {} ".format(num,prime_factors(num)))
```

```
Prime factor of 150 is [2, 3, 5, 5]
```

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 [4]:
```
# take input from the user
n = int(input("Enter value of n: "))
r = int(input("Enter value of r: "))

#implement the formulae of permutations and combinations
fact1 = n
for i in range(1, fact1):
    fact1 = fact1* i


fact2 = n-r
for i in range(1, fact2):
    fact2 = fact2* i

fact3 = r
for i in range(1,fact3):
    fact3 = fact3*i

# permutations
nPr = int(fact1/fact2)
# combinations
nCr = int(nPr/fact3)

print("nPr =",nPr)
print("nCr =",nCr)
```

```
Enter value of n: 10
Enter value of r: 3
nPr = 720
nCr = 120
```

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

In [5]:
```
def DecimalToBinary(num):
    """
    This function converts a decimal number to binary number
    """
    if num ==0:
        return
    else:
        DecimalToBinary(num // 2)
        print(num%2, end="")

DecimalToBinary(9)
```

```
1001
```

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.

In [6]:
```
def cubesum(num):
    """ This function returns the sum of cubes of all the digits in
a given number"""
    n = num
    sum = 0
    while n>0:
        sum += (n%10)*(n%10)*(n%10)
        n //= 10
    return sum


def isArmStrong(num):
    """ This function finds whether the given number is an Armstron
g number or not"""
    if num == cubesum(num):
         return True
    else:
         return False

def printArmStrong(list):
    """This function prints only those numbers that are Armstrong i
n the list"""

    lst = list
    armstrong = []

    for i in lst:
        if isArmStrong(i):
            armstrong.append(i)
    return armstrong

print("The sum of cubes of all the digits in 123 is : ",cubesum(123
))
print("Is 153 an armstrong number: ", isArmStrong(153))
print("Armstrong numbers in the given list : ", printArmStrong([123
,153,0,372,407]))
```

```
The sum of cubes of all the digits in 123 is :  36
Is 153 an armstrong number:  True
Armstrong numbers in the given list :  [153, 0, 407]
```

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

In [7]:
```
def prodDigits(num):
    """ This function returns the product of digits of a given numb
er"""
    n = num
    product = 1
    while n>0:
        digit = n%10
        product*=digit
        n//=10
    return product

num=12345
print("Product of all digits of {} is {}".format(num, prodDigits(nu
m)))
```

```
Product of all digits of 12345 is 120
```

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 [8]:
```
def MDR(num):
    """ This function returns the multiplicative digital root of gi
ven number"""
    s= str(num)
    while len(s)>1:
        s = str(prodDigits(int(s)))
    return int(s)

def MPersistence(num):
    """ This function returns the multiplicative persistance of giv
en number"""
    s= str(num)
    persistence = 0
    while len(s)>1:
        s = str(prodDigits(int(s)))
        persistence+= 1
    return persistence

num= 1234
print("The multiplicative digital root(MDR) of {} is {} " .format(n
um,MDR(num)))
print("The multiplicative persistance of {} is {}".format(num,MPers
istence(num)))
```

```
The multiplicative digital root(MDR) of 1234 is 8
The multiplicative persistance of 1234 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, 18

In [9]:
```
def sumPdivisors(num):
    """This function finds the sum of proper divisors of a given nu
mber"""
    n = num
    divisors =[]
    for i in range(1,n):
        if n%i==0:
            divisors.append(i)
    return sum(divisors)

num=36
print("The sum of proper divisors of {} is {}".format(num,sumPdivis
ors(num)))
```

```
The sum of proper divisors of 36 is 55
```

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 [10]:
```
def perfect(start, end):
    """This function prints all the perfect numbers in a given rang
e"""
    perfect= []
    for i in range(start,end):
        if i == sumPdivisors(i):
            perfect.append(i)
    return perfect

perfect(1,1000)
```

Out[10]:
```
[6, 28, 496]
```

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 [11]:
```
def amicableNumbers(start,end):
    """This function returns pairs of amicable numbers in a range""
"
    pairs = []
    for i in range(start,end+1):
        j = sumPdivisors(i)
        if sumPdivisors(j) == i and i!= j:
            pairs.append(tuple(sorted((i,j))))
    return set(pairs)

amicableNumbers(1,10000)
```

Out[11]:
```
{(220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368
)}
```

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

In [12]:
```
def oddNumber(num):
    """This function returns the odd numbers if num is odd"""
    if num%2==1:
        return num

#create list with numbers from -10 to 10
lst= range(-10,10)
# filter odd numbers in a list by using filter function
oddNumbers = list(filter(oddNumber,lst))

print(oddNumbers)
```

```
[-9, -7, -5, -3, -1, 1, 3, 5, 7, 9]
```

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

In [13]:
```
def cube(num):
    """This function returns the cube of num"""
    return num**3

#create list with numbers from 1 to 10
lst= range(1,10)
# make a list whose elements are cube of elements in a given list b
y using map function
cube = list(map(cube,lst))

print(cube)
```

```
[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 [14]:
```
#create list with numbers from 1 to 10
lst= list(range(1,10))

#filter even numbers in a list by using filter function
even= list(filter(lambda x: (x%2==0),lst))

# make a list whose elements are cube of elements in a given list b
y using map function
cube= list(map(lambda x: (x**3),even))

print("given list:",lst)
print("even numbers in list:",even)
print("cube of even numbers:",cube)
```

```
given list: [1, 2, 3, 4, 5, 6, 7, 8, 9]
even numbers in list: [2, 4, 6, 8]
cube of even numbers: [8, 64, 216, 512]
```