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

In [1]:

```python
def mulTableOfNum(num):
    """
    This function returns a multiplication table for a given num
    """
    for i in range(1,11):
        print("{0} X {1} = {2}".format(num,i,(num*i)))
```

In [2]:

```python
mulTableOfNum(7)
```

```
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70
```

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 [1]:

```python
def isPrime(num):
    """
    This function return a bool value whether the passed argument is either a
    prime number or not
    """
    for i in range(2, num):
        if num % i == 0:

            return False
    return True

def isOddNum(num):
    """
    This function return a bool value whether the passed argument is either odd number
    or not
    """
    if num % 2 != 0:
        return True

    return False

def twinPrimes():
    """
    This funtion returns a array of tuples of consecutive prime numbers
    """
    twinPrimeList = [] #intialise empty list

    #since 2 is even and still prime
    # i am starting from and intilaising the previous num to 3
    previousnum = 3

    for i in range(4,101):
        if isPrime(i) and isOddNum(i):
            #print("({0} ,  {1})".format(previousnum, i))
            #Appending the tuple to twinPrimeList list
            twinPrimeList.append((previousnum, i))

            if i > previousnum:
                #if present num is greater the putting the value to previous value
                previousnum = i


    return twinPrimeList

print(twinPrimes())
```

```
[(3, 5), (5, 7), (7, 11), (11, 13), (13, 17), (17, 19), (19, 23), (23, 29), (29, 31), (31, 37), (3
7, 41), (41, 43), (43, 47), (47, 53), (53, 59), (59, 61), (61, 67), (67, 71), (71, 73), (73, 79),
(79, 83), (83, 89), (89, 97)]
```

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

In [2]:

```
def primeFactor(num):
    """
    This funtion returns a array of prime factors of a number
    """

    primelst = []

    #first we will calculate the number of 2's

    while num % 2 == 0:
        primelst.append(2)#appending all the 2 into the
        num = num / 2

    num = int(num)# converting the float to int again

    for i in range(3,num+1,2):
        while num % i == 0:
            #print("i = {0} and num = {1}".format(i,num))
            primelst.append(i)
            num = num/i

    return primelst

print(primeFactor(56))
```

```
[2, 2, 2, 7]
```

In [3]:

```
print(primeFactor(40))
```

```
[2, 2, 2, 5]
```

In [4]:

```
print(primeFactor(72))
```

```
[2, 2, 2, 3, 3]
```

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 [5]:

```
def product(num):
    """
    This funtion returns a product of num * num-1 * num -2
    """
    product = 1
    for i in range(1,num+1):
        product *= i

    return product

def permutation(n,r):
    """
    This funtion returns the permutation n and r formulae is p(n, r) = n! / (n-r)!.
    """
    nfact = product(n)
    nminusrfact = product(n-r)

    return nfact/nminusrfact


def combination(n,r):
    """
    This funtion returns the permutation n and r formulae is p(n, r) = n! / (n-r)!.
    """
    pnrfact = permutation(n,r)
    rfact = product(r)

    return pnrfact/rfact
```

In [6]:

```
print(permutation(3,2))
```

```
6.0
```

In [7]:

```
print(combination(3,2))
```

```
3.0
```

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

In [8]:

```
def convertDecToBin(num):
    """
    This function return the binary to make we will divide the value by until
    """
    #intial check if 0 then return
    if num == 0:
        return "0"

    #string intialised
    binary = ''

    while num:
        if num & 1 == 1:
            binary = "1" + binary
        else:
            binary = "0" + binary

        #num = int(num)
        num //= 2
    return binary

print(convertDecToBin(10))
```

```
1010
```

In [9]:

```
print(convertDecToBin(4))
```

```
100
```

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 [ ]:

```

```

In [10]:

```
def isArmstrong(num):
    """
    This function checks with the cubeSum() and check if the number is armstrong number
    """
    if cubeSum(num) == num:
        return True
    else:
        return False

def cubeSum(num):
    """
    This function return the sum of cubes of individual digits of the parameter
    """
    temp = num
    #lst = []
    sum1 = 0
    while temp > 0:
        rem = temp % 10  # getting the remimder
        temp //= 10  # getting the quotient
        sum1 += rem ** 3
        #lst.append(rem)
        #print("{0} ----- {1}".format(rem, temp))

    return sum1

def printArmstrong(num):
    """
    This function checks with the isArmstrong() and check if the number is equal to the parameter
    """
    if isArmstrong(num):
        print("{0} is ArmStrong number".format(num))
    else:
        print("{0} is not a  ArmStrong number".format(num))

printArmstrong(153)
printArmstrong(663)
printArmstrong(407)
```

```
153 is ArmStrong number
663 is not a  ArmStrong number
407 is ArmStrong number
```

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

In [11]:

```
def prodDigits(num):
    """
    This function return the product of individual digits of the parameter
    """
    temp = num
    product = 1 #initialising the product to 1 if 0 then the product will always return 0
    while temp > 0:
        rem = temp % 10   # getting the remimder
        temp //= 10   # getting the quotient
        product *= rem

    return product

print("product of 153 is : {0}".format(prodDigits(153)))
print("product of 407 is : {0}".format(prodDigits(407)))
print("product of 663 is : {0}".format(prodDigits(663)))
```

```
product of 153 is : 15
product of 407 is : 0
product of 663 is : 108
```

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 [12]:

```
def computeMDR(num):
    """
    This function returns the MDR of the number
    """
    while True:
        temp = num
        product = 1 #initialising the product to 1 if 0 then the product will always return 0
        while temp > 0:
            rem = temp % 10   # getting the remimder
            temp //= 10   # getting the quotient
            product *= rem #product of

        num = product

        if product < 10:
            break

    return product

print("comp MDR:",computeMDR(341))
```

```
comp MDR: 2
```

In [13]:

```
print("comp MDR:",computeMDR(86))
```

```
comp MDR: 6
```

In [16]:

```
def computeMPersistance(num):
    """
    This function returns the MPersistance of the number
    """
    i = 0
    while True:
        temp = num
        product = 1 #initialising the product to 1 if 0 then the product will always return 0
        while temp > 0:
            rem = temp % 10   # getting the remimder
            temp //= 10   # getting the quotient
            product *= rem #product of

        num = product
        i += 1
        if product < 10:
            break

    return i

print("compute MPersistance :",computeMPersistance(341))
```

```
compute MPersistance : 2
```

In [17]:

```
print("comp MPersistance:",computeMPersistance(86))
```

```
comp MPersistance: 3
```

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 [41]:

```
def sumPdivisors(num):
    """
    This function returns the sum all proper divisors of the number
    """
    sum1 = 0
    lst = []
    for i in range(1,int((num)/2+1)):
        if num % i == 0:
            sum1 += i
            lst.append(i)
#     print(lst)
    return sum1

print(sumPdivisors(36))
```

```
55
```

Note : In the above example 1, 2, 3, 4, 6, 9, 12, 18 are proper divisors

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 [42]:

```
def isPerfectNumber(num):
    """
    This function returns the Boolean, True if is perfect num else False
    """
    if sumPdivisors(num) == num:
        return True
    else:
        return False

print(isPerfectNumber(28))
print(isPerfectNumber(36))
```

```
True
False
```

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 [48]:

```
def areAmicableNum(num1, num2):
    """
    This function returns Boolean, True if amicable else False
    """
    if sumPdivisors(num1) == num2 and sumPdivisors(num2) == num1:
        return True
    else:
        return False

print(areAmicableNum(220,284))
print(areAmicableNum(22, 38))
```

```
True
False
```

In [56]:

```
def lstAmicableNum(num):
    """
    This function returns the list of tuples of amicable pairs between 2 to num+1
    """
    lst = []
    for i in range(2, num + 1):#looping from 2 to num+1
        sPropnum = sumPdivisors(i)# assigning a variable
        #if sPropnum in between 2 to num+1
        if sPropnum <= num+1:
            #check if a<b then return (a,b) else (b,a)
            if sPropnum < i:
                # checking if the amicable
                if areAmicableNum(sPropnum, i):
                    #check if the tuple is contains in lst else append
                    if not ((sPropnum, i) in lst):
                        lst.append((sPropnum, i))

    return lst

print(lstAmicableNum(500))
print(lstAmicableNum(1000))
print(lstAmicableNum(5000))
print(lstAmicableNum(10000))
```

```
[(220, 284)]
[(220, 284)]
```

```
[(220, 284), (1184, 1210), (2620, 2924)]
[(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 [78]:

```
# numbers are from 1 to 100
numbers = list(range(1,101))
```

In [79]:

```
#filtering the odd numbers

lst = list(filter(lambda x: (x%2 != 0), numbers))
```

In [80]:

```
print(lst)
```

```
[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, 5
1, 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 [81]:

```
# numbers are from 1 to 100
numbers = list(range(1,101))
```

In [82]:

```
# using lambda function return the cube of elements in the list
lst = list(map(lambda x : (x**3),numbers))
```

In [83]:

```
print(lst)
```

```
[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, 35937, 39
304, 42875, 46656, 50653, 54872, 59319, 64000, 68921, 74088, 79507, 85184, 91125, 97336, 103823, 1
10592, 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, 571787,
592704, 614125, 636056, 658503, 681472, 704969, 729000, 753571, 778688, 804357, 830584, 857375,
884736, 912673, 941192, 970299, 1000000]
```

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 [84]:

```
# numbers are from 1 to 100
numbers = list(range(1,101))
#filtering the even numbers
numbers = list(filter(lambda x: (x%2) == 0, numbers))
# using lambda function return the cube of elements in the list
lst = list(map(lambda x : (x**3),numbers))
```

In [85]:

```
print(lst)
```

```
[8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832, 8000, 10648, 13824, 17576, 21952, 27000, 32768, 39
304, 46656, 54872, 64000, 74088, 85184, 97336, 110592, 125000, 140608, 157464, 175616, 195112, 216
000, 238328, 262144, 287496, 314432, 343000, 373248, 405224, 438976, 474552, 512000, 551368,
592704, 636056, 681472, 729000, 778688, 830584, 884736, 941192, 1000000]
```