## Question 1

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

In [97]:

```
def print_multiplication_table(number, limit):
    """
    This program takes two inputs; The number whose multiplication table is to be printed and the
limit till where the table
    is to be printed. This program does not return any value to the calling function
    """
    for i in range(limit):
        print("{0} X {1} = {2}".format(number, i, number*i))

#Taking the number and the limit as user inputs|
num = int(input("Enter the number whose table is to be printed:"))
limit = int(input("Enter the limit till where the table is to be printed:"))

print_multiplication_table(num, limit)
```

```
Enter the number whose table is to be printed:9
Enter the limit till where the table is to be printed:15
9 X 0 = 0
9 X 1 = 9
9 X 2 = 18
9 X 3 = 27
9 X 4 = 36
9 X 5 = 45
9 X 6 = 54
9 X 7 = 63
9 X 8 = 72
9 X 9 = 81
9 X 10 = 90
9 X 11 = 99
9 X 12 = 108
9 X 13 = 117
9 X 14 = 126
```

## Question 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 [98]:

```
"""
One method is using a for loop and iterating over all numbers. Another method is to store all prim
e numbers under 1000
in an array and checking the difference of consecutive elements. Using Sieve of Eratosthenes to ge
t the array of primes.
"""

def get_sieve_list():
    """
    This function returns a list containing all prime numbers under 1000
    """
    # List is initialized to ones initially.
    sieve_list = [1 for i in range(1001)]
    sieve_list[0] = 0
    sieve_list[1] = 0
    i = 2
    # Iterating from 2 till number whose square is less than or equal to 1000
    while(i**2 <= 1000):
        # If sieve_list[i] is 1, that means it's value hasn't been modified meaning it's a prime nu
mber
        if(sieve_list[i] == 1):
            # Converting all multiples of i to 0, indicating that they are not prime numbers
            for j in range(i*2, 1001, i):
```

```
                sieve_list[j] = 0
        i += 1
    # Creating an empty list that will contain prime numbers
    primes_list = []
    # Adding the indices of elements whose value is 1, i.e., prime
    for i in range(1000):
        if(sieve_list[i] == 1):
            primes_list.append(i)
    # Returning the prime numbers list
    return primes_list

primes_list = get_sieve_list()

# Creating an empty twin_primes list that will contain a list of tuples where each tuple is a set
of twin primes.
twin_primes = []

# Iterating over the primes_list list looking for twin primes
for i in range(1, len(primes_list)-1):
    if(primes_list[i+1] - primes_list[i] == 2):
        twin_primes.append(tuple([primes_list[i], primes_list[i+1]]))

# Printing the twin primes
print("The twin primes under 1000 are:")
for twin_prime in twin_primes:
    print(twin_prime)
```

```
The twin primes under 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)
```

## Question 3

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

In [99]:

```
import math

def print_prime_factors(num):
    """
```

```
    This program takes a number as an input and prints it's prime factors
    """
    # Empty list to which prime factors will be appended
    prime_factors = []
    # If the number is even, keep dividing it with 2 until it becomes odd while appending 2 to the
list every iteration
    while num%2 == 0:
        prime_factors.append(2)
        num /= 2
    # Looking for odd numbers that divide the number till sqrt(number). Adding the number if it di
vides the number to the list
    for i in range(3, int(math.sqrt(num))+1, 2):
        while(num%i == 0):
            prime_factors.append(int(i))
            num /= i
    # At this point if the number is still greater than 2, then it means the number is a prime
number itself.
    # Appending the remaining value to the list
    if num > 2:
        prime_factors.append(int(num))
    return prime_factors

# Taking user input
num = int(input("Enter the number whose prime factors are to be printed:"))
print("The prime factors of the number are:", print_prime_factors(num))



#References:
#1. https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/
```

```
Enter the number whose prime factors are to be printed:56
The prime factors of the number are: [2, 2, 2, 7]
```

## Question 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 [100]:

```
def fact(n):
    """
    This function takes a value as an input and returns it's factorial
    """
    if n==0 or n==1:
        return 1
    else:
        return n * fact(n-1)

# Taking user input for the values n and r
n, r = [int(x) for x in input("Enter the values of n and r seperated by a space: ").split()]

#Printing the values
print("The number of permutations of n objects taken r at a time are: " + str(int(fact(n)/fact(n-r)
)))
print("The number of combinations of n onjects taken r at a time are: " + str(int(fact(n)/(fact(r)*
fact(n-r)))))
```

```
Enter the values of n and r seperated by a space: 10 2
The number of permutations of n objects taken r at a time are: 90
The number of combinations of n onjects taken r at a time are: 45
```

## Question 5

**Write a function that converts a decimal number to binary number**

In [101]:

```
def print_binary(num):
    """
    This function takes the input in decimal format and returns a string in binary format
    by repeatedly dividing the number by 2 and adding the remainder to the end of the
    string.
    """
    # Handling boundary case
    if num == 0:
      return str(0)
    bin_n = ""
    while(num != 0):
      bin_n += str(num%2)
      num //= 2
    return bin_n[::-1]

# Taking the input in decimal
num = int(input("Enter the number in decimal format: "))
print("The Binary representation of the given number is: " + print_binary(num))
```

```
Enter the number in decimal format: 13
The Binary representation of the given number is: 1101
```

## Question 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 [102]:

```
def cubesum(num):
  """
  This function takes an integer as an input and returns it's cubesum.
  It iterates over the integer by temporarily typecasting it as a string
  and calculates it's cube and adds it to the cubesum variable which is
  initialized to 0
  """
  cubesum = 0
  for i in range(len(str(num))):
    cubesum += int(str(num)[i])**3
  return cubesum

def PrintArmstrong(limit):
  """
  This function takes a number as an input and prints the armstrong
  numbers till a number
  """
  for i in range(limit):
    if(i == cubesum(i)):
      print(i, end=" ")

def isArmstrong(num):
  """
  This function takes a number as an input and returns true if it is
  an Armstrong number or returns False if it isn't an Armstrong number.
  """
  if num == cubesum(num):
    return True
  else:
    return False

# Taking user inputs
limit = int(input("Enter the limit till where the Armstrong numbers are to be printed: "))
print("The Armstrong numbers till " + str(limit) + " are: ")
PrintArmstrong(limit)
num = int(input("\nEnter the number that is to be checked for Armstrong property: "))
print("Given number is an Armstrong number") if isArmstrong(num) else print("Given number is not
an Armstrong number")
```

```
Enter the limit till where the Armstrong numbers are to be printed: 500
The Armstrong numbers till 500 are:
0 1 153 370 371 407
Enter the number that is to be checked for Armstrong property: 371
```

```
Given number is an Armstrong number
```

## Question 7

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

In [103]:

```
import functools

def prodDigits(num):
  """
  This function takes a number as an input and returns the product of digits of that number by usi
ng reduce function.
  """
  product = 0
  product = functools.reduce(lambda x, y: int(x) * int(y), list(str(num)))
  return product

# Taking user input
num = int(input("Enter the number: "))
print("The product of the digits of the number is: " + str(prodDigits(num)))
```

```
Enter the number: 56
The product of the digits of the number is: 30
```

## Question 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 andmultiplicative persistence respectively**

In [104]:

```
import functools

def prodDigits(num):
  """
  This function takes a number as an input and returns the product of digits of that number by usi
ng reduce function.
  """
  product = 0
  product = functools.reduce(lambda x, y: int(x) * int(y), list(str(num)))
  return product

def MDR(num):
  """
  This function takes a number as an input and returns the Multiplicative Digital Root of the numb
er using recursion.
  """
  if num < 10:
    return num
  else:
    return MDR(prodDigits(num))

def MPersistence(num):
  """
```

```
  This function takes a number as an input and returns the MPeristence of the number using recursi
on.
  """
  if num < 10:
    return 0
  else:
    return 1 + MPersistence(prodDigits(num))

num = int(input("Enter the number for which MDR and MPersistence is to be found: "))
print("The Multiplicative Digital Root of the number {0} is {1} and it's MPersistence is {2}".form
at(num, MDR(num), MPersistence(num)))
```

```
Enter the number for which MDR and MPersistence is to be found: 341
The Multiplicative Digital Root of the number 341 is 2 and it's MPersistence is 2
```

## Question 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 [105]:

```
def sumPdivisors(num):
  """
  This function return the sum of the proper divisors of the given number using iteration to itera
te over numbers less than the number.
  """
  sum = 0
  for i in range(1, num//2+1):
    if(num%i == 0):
      sum += i
  return sum

#Taking user input
num = int(input("Enter the number who's sum of proper divisors is to be found: "))
print("The sum of proper divisors of the given number is {0}".format(sumPdivisors(num)))
```

```
Enter the number who's sum of proper divisors is to be found: 36
The sum of proper divisors of the given number is 55
```

## Question 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 [106]:

```
def sumPdivisors(num):
  """
  This function return the sum of the proper divisors of the given number using iteration to itera
te over numbers less than the given number.
  """
  sum = 0
  for i in range(1, num//2+1):
    if(num%i == 0):
      sum += i
  return sum

# Taking user input
lower_limit, upper_limit = [int(x) for x in input("Enter the lower and upper limits of the range to
find the perfect numbers: ").split()]
```

```
print("Perfect numbers in the given range are: ")

for num in range(lower_limit, upper_limit+1):
  if(num == sumPdivisors(num)):
    print(num)
```

```
Enter the lower and upper limits of the range to find the perfect numbers: 1 100
Perfect numbers in the given range are:
6
28
```

## Question 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.**

**Write a function to print pairs of amicable numbers in a range**

In [107]:

```
def sumPdivisors(num):
  """
  This function return the sum of the proper divisors of the given number using iteration to itera
te over numbers less than the given number.
  """
  sum = 0
  for i in range(1, num//2+1):
    if(num%i == 0):
      sum += i
  return sum

# Taking user input
lower_limit, upper_limit = [int(x) for x in input("Enter the lower and upper limits of the range to
find the paris of amicable numbers: ").split()]

print("Amicable pairs in the given range are: ")

# Creating a list that store amicable numbers to prevent repitions of the same pair
amicable_numbers = []

for i in range(lower_limit, upper_limit+1):
  temp = sumPdivisors(i)
  if(sumPdivisors(temp) == i and temp <= upper_limit):
    if i not in amicable_numbers and temp not in amicable_numbers:
      print(tuple([i, temp]))
      amicable_numbers.extend([i, temp])
```

```
Enter the lower and upper limits of the range to find the paris of amicable numbers: 200 300
Amicable pairs in the given range are:
(220, 284)
```

## Question 12

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

In [108]:

```
# Taking the list as an input from the user
nums = [int(x) for x in input("Enter the elements of the list").split()]
#Using lambda functinos to filter out the even numbers and storing them in a list called odd_nums
odd_nums = list(filter(lambda x: x%2 != 0, nums))
print("The odd numbers in the entered list are: ")
print(odd_nums)
```

```
Enter the elements of the list1 2 3 9 8 7
The odd numbers in the entered list are:
[1, 3, 9, 7]
```

## Question 13

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

In [109]:

```
# Taking the list as an input from the user
nums = [int(x) for x in input("Enter the elements of the list").split()]
# Using lambda functions and map to store cubes in a list
nums_cubes = list(map(lambda x: x**3, nums))
print("The cubes of the given numbers are: ")
print(nums_cubes)
```

```
Enter the elements of the list1 2 3 9 8 7
The cubes of the given numbers are:
[1, 8, 27, 729, 512, 343]
```

## Question 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 [110]:

```
# Taking the list as an input from the user
nums = [int(x) for x in input("Enter the elements of the list: ").split()]
# Using filter to filter out the odd numbers and passing them to map as an iterable to get the cub
es of all even numbers
nums_even_cubes = list(map(lambda x: x**3, list(filter(lambda x: x%2 == 0, nums))))
print("The cubes of all even numbers in the given list are: ")
print(nums_even_cubes)
```

```
Enter the elements of the list: 1 2 3 9 8 7
The cubes of all even numbers in the given list are:
[8, 512]
```