In [120]:

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

In [121]:

```
def multiplicationTable(num):
    '''
    Function that takes input of a number and prints multiplication of a number
    '''
    for i in range(1 , 11):
        print("{0} * {1}  = {2}".format(num , i , num*i))
```

In [122]:

```
multiplicationTable(12)
```

```
12 * 1  = 12
12 * 2  = 24
12 * 3  = 36
12 * 4  = 48
12 * 5  = 60
12 * 6  = 72
12 * 7  = 84
12 * 8  = 96
12 * 9  = 108
12 * 10  = 120
```

In [ ]:

```
## Write a program to print twin primes less than 1000. If two consecutive odd numbers are both pr
ime then they are known
## as twin primes
```

In [82]:

```
def getPrimes(num):
    for i in range(2,num):
        if num%i == 0:
            return False
    return True

def get_twinPrimes(num):
    twinPrime = []
    lst = [i for i in range(2,num) if getPrimes(i)]
    for i in range(len(lst)-1):
        if lst[i+1] - lst[i] == 2:
            twinPrime.append((lst[i],lst[i+1]))
    return twinPrime
```

In [123]:

```
get_twinPrimes(1000)
```

Out[123]:

```
[(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)]
```

In [124]:

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

In [84]:

```
import math
def GetPrimeFactor(num):
    print(1)
    while num%2 == 0:
        print(2)
        num=num/2

    for i in range(3 , int(math.sqrt(num))):
        while num%i == 0:
            print(num)
            num=num/i

    if num > 2:
        print(num)
```

In [85]:

```
GetPrimeFactor(77)
```

```
1
77
11.0
```

In [86]:

```
##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 [87]:

```
def factorial(num):
    if num == 1:
        return 1
    else:
        return (num * factorial(num-1))

def Permutations( n , r):
    return factorial(n) / factorial(n-r)

def combinations(n , r):
    return (Permutations(n,r) / factorial(r))
```

In [88]:

```
## Write a function that converts a decimal number to binary number
```

In [89]:

```
def binNum(num):
    lst = []
    while num > 1:
        tupl = divmod(num,2)
        lst.append(tupl[1])
        num = tupl[0]
    lst.append(tupl[0])
    lst.reverse()
    print(*lst,sep='')
```

In [90]:

```
lst = binNum(32)
```

```
100000
```

In [91]:

```
# 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 [92]:

```
 def no_ofDigits(num):
    count=0
    while num > 0:
        num = num // 10
        count = count+1
    return count

def cubesum(num):
    sum=0
    num_len = no_ofDigits(num)
    while num > 0 :
        digit = num % 10
        sum = sum + digit ** num_len
        num = num // 10
    return sum

def isArmstrong(num):
    if num == cubesum(num):
        return True
    else:
        return False

def PrintArmstrong(num):
    if isArmstrong(num):
        print('{} is an Armstrong Number'.format(num))
    else:
        print('{} is not an Armstrong number'.format(num))
```

In [93]:

```
PrintArmstrong(153)
```

```
153 is an Armstrong Number
```

In [94]:

```
PrintArmstrong(101)
```

```
101 is not an Armstrong number
```

In [95]:

```
# Write a function prodDigits() that inputs a number and returns the product of digits of that num
ber
```

In [96]:

```
def prodDigits(num):
    rem = 1
    while num > 1:
        rem = rem * (num%10)
        num = num // 10
    return rem
```

In [97]:

```
prodDigits(1234)
```

Out[97]:

```
24
```

In [98]:

```
# 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 [99]:

```
def MDR(num):
    count = 0
    while no_ofDigits(num) > 1 :
        num = prodDigits(num)
        count= count+1
    print ('MDR {} , MPersistence {}'.format(num,count))
```

In [100]:

```
MDR(341)
```

```
MDR 2 , MPersistence 2
```

In [101]:

```
# 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 [102]:

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

In [103]:

```
sumPdivisors(36)
```

Out[103]:

```
55
```

In [104]:

```
# 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 [105]:

```
def PerfectSum(lrange , hrange):
    for i in range(lrange , hrange):
        if i == sumPdivisors(i):
            print(i,end=' ')
```

In [106]:

```
PerfectSum(20 , 1000)
```

```
28 496
```

In [107]:

```
# 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 [108]:

```
def amicableNum(lrange , hrange):
    numbers = []
    lst = range(lrange , hrange)
    for i in lst:
        num = sumPdivisors(i)
        if (num in lst and sumPdivisors(num) in lst) and ((num,i) not in numbers):
            tupl = (i,num)
            numbers.append(tupl)
    return numbers
```

In [109]:

```
amicableNum(200,300)
```

Out[109]:

```
[(220, 284), (246, 258), (248, 232), (268, 208), (272, 286), (290, 250)]
```

In [110]:

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

In [111]:

```
def oddNum(lst):
    return list(filter(lambda x : x%2 != 0 ,lst))
```

In [112]:

```
oddNum([1,2,3,4,5,6,7,8,9,10])
oddNum([1,2,3,4,5,6,7,8,9,10])
```

Out[112]:

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

In [113]:

```
# Write a program which can map() to make a list whose elements are cube of elements in a given li
st
```

In [114]:

```
def cubeOfNum(x):
    return list(map(lambda x: x** 3 , x))
```

In [115]:

```
x = [1 ,2 , 3 , 4 , 5 , 6, 7, 8]
```

In [116]:

```
cubeOfNum(x)
```

Out[116]:

```
[1, 8, 27, 64, 125, 216, 343, 512]
```

In [117]:

```
# Write a program which can map() and filter() to make a list whose elements are cube of even numb
er in a given list
```

In [118]:

```
def cubeOfEven(lst):
    x = list(filter(lambda x : x%2 == 0 , lst))
    return list(map(lambda x : x ** 3 , x))
```

In [119]:

```
cubeOfEven([1,2,3,4,5,6,7,8,9,10])
```

Out[119]:

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