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

In [2]:
```
# function definition
def mulTable(number):
    # below is docstring
    """prints the multiplication table of given number"""

    for i in range(1,11):
        print("{n} * {i} = {r}".format(n=number,i=i,r=number*i))
```

In [3]:
```
# Calling the mulTable function
mulTable(19)
```

```
19 * 1 = 19
19 * 2 = 38
19 * 3 = 57
19 * 4 = 76
19 * 5 = 95
19 * 6 = 114
19 * 7 = 133
19 * 8 = 152
19 * 9 = 171
19 * 10 = 190
```

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 [4]:
```
# defining the isConsecutive function to check two given numbers is consecutive or not
def isConsecutive(n1, n2):
    '''this function check that given two prime number is consecutive or not.
        It returns 1 if given numbers are consecutive else returns 0.'''
    result1 = isPrime(n1)
    result2 = isPrime(n2)

    if result1 and result2:
        return 1
    else:
        return 0


# defining the idPrime function to check given number is prime or not
def isPrime(number):
    '''this function check whether given number is prime or not.
        It returns 1 if given number is prime else returns 0.'''
    flag = 0
    for i in range(2,number):
        if number%i == 0:
            flag = 1
    if flag== 0:
        return 1
    else:
        return 0

for i in range(3,998,2):
    # range is started by 3(not 2) because 2 is even-prime
    # and we need odd primes
    # range end at 997 b'z we need prime numbers less than 1000

    # assign two consecutive odd number
    num1, num2 = i, i+2

    if isConsecutive(num1, num2):
        print("%d  %d"%(num1,num2))
```

```
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 [5]:
```
# helped by: stackoverflow to understand the steps to find prime factors of given number
def primeFactor(number):
    '''this function returns the list of prime factors'''
    p_factor=[] # initializing the list to store the prime factors
    i = 2 # initializing iteration variable

    while i**2 <= number:
    # iterate the loop till i^2 <= number.

        if number%i!=0:
            # check whether it is divisible by iterator(i) or not,
            # if not divisible then increment the iterator(i)
            i += 1
        else:
            # pointer(means program flow,not "C" pointer) is in else means
            # number is divisible by iterator(i) means
            # iterator is prime factor

            # so divide number by the prime factor
            number //=i

            # finally, append to the list of prime factor 'p_factor'
            p_factor.append(i)
    if number>1:
        # after all processing if n is greater than 1 then append it to prime factor
        p_factor.append(number)

    # finally, return p_factor(A list containing all prime factors)
    return p_factor
```

In [6]:
```
# let's test primeFactor
primeFactor(3453234)
```

Out[6]:
```
[2, 3, 373, 1543]
```

Question 4: Write a function that converts a decimal number to binary number.

In [13]:
```
# I know Sir, that my strategy might be different(and even worst).
# But, after a long time for searching over google(I didn't find any one solution)
# over google every writer were just printing the binary number
# printing the binary number were really easy
# But, I find difficulty to write a function
# and finally.....................
# I come to this conclusion



def decimalToBinary(dec_number):
    '''this function returns only binary of positive number'''

     # initializing the string to concatenate the remainder into string
    binary_number = ''

    # check whether given number is less than 0('zero'), If YES... return 0.
    if dec_number <= 0:
        return 0;

    # CRUX part(hahaha...)
    while(dec_number>0):
        # iterate loop till decimal number is greater than 0

        # compute remainder and store it in 'rem' variable
        rem = dec_number % 2

        # divide the number by 2 to use this processed number for next iteration
        dec_number //= 2

        # concatenate the remainder with string(binary_number)
        binary_number += str(rem)


    # step1: reverse the string
    # step2: typecast the string(binary_number) to an integer number
    # step3: override the binary_number variable by new processed binary number
    binary_number= int(binary_number[::-1])

    # finally return the number
    return binary_number
```

In [14]:
```
decimalToBinary(92)
```

Out[14]:
```
1011100
```

In [15]:
```
# let's upgrade the function to support -ve(negative) numbers binary also

def decimalToBinaryUpgraded(number):
    '''In case of negative number it will return string not integer binary value'''
    if number == 0:
        return 0

    elif number > 0:
        binary_of_number = decimalToBinary(number)
        return binary_of_number

    elif number < 0:

        number = abs(number)
        binary_of_number = decimalToBinary(number)

        # store the binary of given number as string in temp
        temp = str(binary_of_number)

        #let's reverse the 'temp' variable to implement the idea of 2's complement
        temp = temp[::-1] # the fastest way to reverse the string

        #let's initialize the count value by 0
        count = 0

        new_String = ''
        for c in temp:
            if c=='0' and count == 0:
                new_String += '0'
            elif c=='1' and count == 0:
                count = 1
                new_String += '1'
            elif c=='0' and count == 1:
                new_String += '1'
            elif c=='1' and count == 1:
                new_String += '0'

        return (new_String[::-1])
```

In [16]:
```
# let's test decimalToBinaryUpgraded
decimalToBinaryUpgraded(66)
```

Out[16]: 1000010

In [17]:
```
decimalToBinaryUpgraded(-10)
# In case of negative number string is returned because
# when we typecast the given string into int, the 'zero' (first digit of binary removed)
```

Out[17]: '0110'

Question 5: 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)!

combinations of n objects taken r at a time is

c(n,r) = n!/(r!*(n-r)! .

In [18]:
```
# let's write function to find factorial first

def factorial(n):
    '''this function returns the factorial of given number.'''
    if n==0:
        return 0
    elif n<0:
        print('oops! You entered wrong number')
        exit(0)
    else:
        result=1
        for i in range(1,n+1):
            result = result*i
        return result

factorial(5)
```

Out[18]: 120

In [19]:
```
# let's write a function to find permutation

def permutations(n,r):
    '''this function returns permutation.'''
    return factorial(n)/factorial(n-2)

permutations(5,2)
```

Out[19]: 20.0

In [20]:
```
# let's write a function to find combination

def combinations(n,r):
    '''this functions returns combination'''
    return factorial(n)/(factorial(r)*factorial(n-r))

combinations(5,2)
```

Out[20]: 10.0

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.

Armstrong number: is a number that is equal to the sum of cubes of it's every digit for ex: 153: 1^3 + 5^3 + 3^3 => 1+125+27 = 153, right.

In [21]:
```
# function definition of 'cubesum()'

def cubesum(number):
    '''this function returns the sum of cube of digits of given number.'''
    #digit is to store the digits of the given number
    digit = 0

    # result variable is to store the sum of cubes
    result = 0

    # iterate the loop till number>0
    while(number>0):

        # finding the digit by applying remainder by 10 to the given number
        digit = number%10

        # storing the result into result variable
        result += digit**3

        # divide number by 10 for next iteration because I have to find next digit
        number //= 10

    # finally retur the calculated value
    return result
```

In [22]:
```
cubesum(153)
```

Out[22]: 153

In [23]:
```
def printArmstrong(initial,final):
    '''this method prints the armstrong number between the given range.

        'initial':

             dtype: int(integer).
            'initial' >=1 ( initial should be greater than or equal to 1)
            'initial' will be included in the range

        'final':
             dtype: int(integer).
             final <= 999

            'final' can be any Natural number.
            'final' will also be included in the range.
    '''
    print("Armstrong numbers between %d and %d"%(initial,final))
    for i in range(initial,final+1):
        if i == cubesum(i):
            print(i)

printArmstrong(1,999)
```

```
Armstrong numbers between 1 and 999
1
153
370
371
407
```

In [26]:
```
def isArmstrong(number):
    if number == cubesum(number):
        return 1
    else:
        return 0

result = isArmstrong(153)

if result:
    print('Yes')
else:
    print('No')
```

```
Yes
```

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

In [27]:
```
# definition of prodDigits()

def prodDigits(number):
    '''this function returns the product of digits of given number.'''
    if number==0:
        return 0

    product = 1

    while(number>0):
        digit = number%10
        product *= digit
        number //= 10

    return product
```

In [28]:
```
prodDigits(12345)
```

Out[28]:
```
120
```

In [29]:
```
prodDigits(10234)
```

Out[29]: 0

Question 8: If all digits of a 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.

In [40]:
```
# defining multiplicativeDigitalRoot function

def MDR_and_persistance(number):
    mdr = number
    count =0
    while(mdr > 9):
        count +=1
        mdr = prodDigits(mdr)

    # return mdr and persistance
    return mdr,count
```

In [41]:
```
mdr,persistance = MDR_and_persistance(23487)
```

In [42]:
```
mdr, persistance
```

Out[42]: (6, 4)

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 [68]:
```
# defining sumPdivisors()

def sumPdivisors(number):
    result=0
    divisors=[]
    for i in range(1,number):
        if number%i==0:
            result += i
            divisors.append(i)
    return divisors,result
```

In [69]:
```
sumPdivisors(2)
```

Out[69]: ([1], 1)

In [70]:
```
sumPdivisors(6)
```

Out[70]: ([1, 2, 3], 6)

In [71]:
```
sumPdivisors(12)
```

Out[71]: ([1, 2, 3, 4, 6], 16)

In [54]:
```
sumPdivisors(15)
```

Out[54]: ([1, 3, 5], 9)

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 [55]:
```
# function to return all perfect number in given range.

def perfectNumber(first,last):
    result=[]
    for i in range(first,last+1):
        _,div_sum = sumPdivisors(i)

        if div_sum == i:
            result.append(i)

    return result
```

In [56]:
```
perfectNumber(1,10000)
```

Out[56]: [6, 28, 496, 8128]

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.
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 [65]:
```
def amicablePair(first,last):

    result=[]
    for i in range(first,last+1):
        _, div_sum_i= sumPdivisors(i)
        _, div_sum_1= sumPdivisors(div_sum_i)

        if div_sum_1 == i:
                result.append((i,div_sum_i))

    return result
```

In [67]:
```
amicablePair(1,10000)
```

Out[67]:
```
[(6, 6),
 (28, 28),
 (220, 284),
 (284, 220),
 (496, 496),
 (1184, 1210),
 (1210, 1184),
 (2620, 2924),
 (2924, 2620),
 (5020, 5564),
 (5564, 5020),
 (6232, 6368),
 (6368, 6232),
 (8128, 8128)]
```

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

In [11]:
```
# let's create a function which returns filtered odd numbers from a given list

def odd_filter(number):

    # if given number is odd then return otherwise, do nothing
    if number%2 !=0:
        return number

#list of numbers
numbers = [23,45,34,45,6,67,5,6,54,56,5,454,45,45,5,6,7,7,676]

# filtering odd number from the list using filter method
odd_numbers = filter(odd_filter,numbers)
```

In [12]:
```
list(odd_numbers)
```

Out[12]:
```
[23, 45, 45, 67, 5, 5, 45, 45, 5, 7, 7]
```

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

In [3]:
```
# let's write a function

def cube(item):
    return item**3

list1=[1,2,3,4,5,6]

list2 = list(map(cube,list1))


print('list1 :',list1)
print('list2 (after perform cube operation on list):',list2)
```

```
list1 : [1, 2, 3, 4, 5, 6]
list2 (after perform cube operation on list): [1, 8, 27, 64, 125, 216]
```

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 [1]:
```
def cube(item):
    return item**3

def even_number(item):
    if item%2 ==0:
        return item
```

In [3]:
```
num_list = list(range(10))
```

In [4]:
```
num_list
```

Out[4]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [7]:
```
cube_list = list(map(cube,num_list))
```

In [8]:
```
cube_list
```

Out[8]:
```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

In [10]:
```
cube_even_list = list(filter(even_number,cube_list))
```

In [11]:
```
cube_even_list
```

Out[11]:
```
[8, 64, 216, 512]
```

In [ ]:
```

```