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

In [1]:
```
def get_multiplication_table(val,table_till):
    #print(val)
    #print(table_till)
    """This function outputs the multiplication table"""

    for i in range(1,table_till+1):
        print (" {0} X {1} = {2}".format(val,i,val*i))

num = int(input("Please enter the number you want the table of : " ))
final_res = int(input("Please enter the number till which you want to compute the table

get_multiplication_table(num,final_res)
#print(get_multiplication_table.__doc__)
```

```
Please enter the number you want the table of : 1
Please enter the number till which you want to compute the table : 4
 1 X 1 = 1
 1 X 2 = 2
 1 X 3 = 3
 1 X 4 = 4
```

1. 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]:
```
primes = []

num = 100
for num in range(2,num+1):
    isPrime = True
    for i in range(2,num):
        if num % i == 0:
            isPrime = False
    if isPrime == True:
        primes.append(num)
#print (primes)

for j in range(0,len(primes)):
    if primes[j] - primes[j-1] == 2:
        print(primes[j-1],primes[j])
```

```
3 5
5 7
11 13
17 19
29 31
41 43
59 61
71 73
```

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 isprime(n):
    isprime_status=True
    for i in range(2,n):
        if n % i == 0:
            isprime_status = False
```

```
            break
        else:
            isprime_status = True
            break
    return isprime_status

def fact(num):
    facts = []
    for i in range(2,num):
        if num % i == 0:
            facts.append(i)
    return facts

def primefact(final_num):
    new_lst = fact(final_num)
    #print(new_lst)

    final_lst = []

    for k in new_lst:
        if isprime(k) == True:
            final_lst.append(k)

    print(final_lst)

primefact(108)
```

```
[2, 3, 9, 27]
```

1. 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 [134…

```
def get_factorial(a):
    fact = 1
    for i in range(1,a+1):
        fact = fact * i
    return fact

#get_factorial(4)

def permutation(n,r):

    per = (get_factorial(n))/(get_factorial(n-r))
    return per

def combination(n,r):

    comb = (permutation(n,r))/(get_factorial(r))
    return comb

print("Permutation is " , permutation(30,4))

print("Combination is " , combination(30,4))
```

```
Permutation is  657720.0
Combination is  27405.0
```

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

In [23]:
```
def to_string(a):
    return str(a)


def get_binary(num):
    binary = []
    while num != 1:
        binary.append(num % 2)
        num = int(num / 2)
    binary.append(1)
    binary.reverse()
    return binary

num = int(input("Enter a number : "))
#get_binary(num)

final_lst = list(map(to_string , get_binary(num)))
#print(final_lst)

Bin_number = ''.join(final_lst)
print(Bin_number)
```

```
Enter a number : 17
10001
```

1. 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 [27]:
```
def cubesum(a):
    sum1 = 0
    string_a = str(a)

    for i in range(len(string_a)):
        sum1 += (int(string_a[i])*int(string_a[i])*int(string_a[i]))
    return sum1

def isArm(b):
    if b== cubesum(b):
        print("Armstrong")
    else:
        print("Not Armstrong")




#cubesum(123)
isArm(153)
```

```
Armstrong
```

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

In [51]:
```
def prodDigits(num1):
    str_num1 = str(num1)
    prod =1
    for i in range(len(str_num1)):
```

```
        prod *= int(str_num1[i])
    return prod

prodDigits(333)
```

Out[51]: 27

1. 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 [35]:
```
def prodDigits(num1):
    str_num1 = str(num1)
    prod =1
    for i in range(len(str_num1)):
        prod *= int(str_num1[i])
    return prod

def MDR(x):
    while True:

        x = prodDigits(x)
        if len(str(x)) == 1:
            return x

def MPersistence(x):
    cnt = 0
    while True:
        cnt +=1
        x = prodDigits(x)
        if len(str(x)) == 1:
            return cnt

num = 341

print("MDR : {} , MPersistence : {}".format(MDR(num),MPersistence(num)))
```

```
MDR : 2 , MPersistence : 2
```

1. 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(x):
    pdiv = []
    for i in range(1,x):
        if x%i == 0:
            pdiv.append(i)
    return pdiv

print (sumPdivisors(36))
```

```
[1, 2, 3, 4, 6, 9, 12, 18]
```

1. 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 [22]:
```
def perfect_num(x):
    pdiv = []
    for i in range(1,x):
        if x%i == 0:
            pdiv.append(i)
    if sum(pdiv) == x:
        return ("perfect")

r = 10000
p_num = []

for j in range(1,r):
    if perfect_num(j) == "perfect":
        p_num.append(j)

print (p_num)
```

```
[6, 28, 496, 8128]
```

1. 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 [71]:
```
def propDivisor(num1):
    lst = []
    for i in range(1,num1):
        if num1%i==0:
            lst.append(i)
    return sum(lst)



def getAmicable(r):

    for i in range(1,r):
        #print ("i : " , i)
        for j in range(1,i+1):
            if propDivisor(i) == j and propDivisor(j) == i:
                if i!= j:
                    print(i,j)
            #print ("j : " , j)
getAmicable(2000)
```

```
284 220
1210 1184
```

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

In [4]:
```
def odd_num(x):

    if(x%2) == 0:
        return False
    else:
```

```
        return True

lst = [3,4,5,6,7,8,9]

odd_numbers = filter(odd_num,lst)

for num in odd_numbers:
    print (num)
```

```
3
5
7
9
```

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

In [5]:
```
def cube_num(x):
    return x**3

lst = [3,4,5,6,7,8,9]

cube_numbers = map(cube_num,lst)

for num in cube_numbers:
    print (num)
```

```
27
64
125
216
343
512
729
```

1. Write a program which can map() and filter() to make a list whose elements are cube of even number in a given list

In [6]:
```
def odd_num(x):

    if(x%2) == 0:
        return True
    else:
        return False

def cube_num(x):
    return x**3

lst = [3,4,5,6,7,8,9]

cube_even_numbers = map(cube_num,filter(odd_num,lst))

for num in cube_even_numbers:
    print (num)
```

```
64
216
512
```