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

In [12]:
```
# function takes num as input and print its table
def input_num_print_table(num):
    result = 0
    for i in range(1, 11):
      result = result + num
      print (num, "*", i, "=", result)

input_num_print_table(11)
```

```
11 * 1 = 11
11 * 2 = 22
11 * 3 = 33
11 * 4 = 44
11 * 5 = 55
11 * 6 = 66
11 * 7 = 77
11 * 8 = 88
11 * 9 = 99
11 * 10 = 110
```

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 [61]:

```
import math

# function takes n an input and return boolean list with prime numbers
def seive_of_eratosthenes(n):

  prime = [True for x in range(n + 1)]
  prime[0], prime[1] = False, False
  i = 2
  for i in range(2, int(math.sqrt(n))+1):
    if prime[i] == True:
      for j in range(i * 2, n + 1, i):
        prime[j] = False

  return prime

# function takes end as input and print twin prime numbers less than e
nd
def print_twin_primes(end):
  prime = seive_of_eratosthenes(end)
  n = len(prime)
  for i in range(1, n, 2):
    if prime[i] == True and i+2 < n and prime[i+2] == True:
      print("({0}, {1})".format(i, i+2))

print_twin_primes(1000)
```

```
(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)
```

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

In [1]:
```
# function takes number as input and print its prime factors
def print_prime_factors(number):
  i = 2
  print ("Prime factors of {0} are:".format(number))
  while i <= number:
    if number % i == 0:
      print (i)
      number = number/i
      i = 2
    else:
      i += 1

number = int(input("Enter a number "))
print_prime_factors(number)
```

```
Enter a number 21
Prime factors of 21 are:
3
7
```

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 [27]:
```
# function takes num as input and return its factorial
def factorial(num):
  if num < 0:
    return None
  res = 1
  for i in range(1, num+1):
    res *= i
  return res

# function returns number od permutation of n objects taken r at a time
def permutation(n, r):
  if n < r or n < 0 or r < 0 or n < r:
    return None
  return int(factorial(n) / factorial(n-r))

# function returns number of combinations of n objects taken r at a time
def combination(n, r):
  permute = permutation(n, r)
  return int(permute / factorial(r))

n, r = 6, 3
print ("Permutation of {0} objects taking {1} at a time:".format(n, r), permutation(n, r))
print ("Combination of {0} objects choosing {1} at a time:".format(n, r), combination(n, r))
```

```
Permutation of 6 objects taking 3 at a time: 120
Combination of 6 objects choosing 3 at a time: 20
```

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

In [ ]:
```
# function takes decimal number dec and return its binary
def decimal_to_binary(dec):
  result = ''
  while dec >= 1:
    r = dec % 2
    dec = dec // 2
    result = str(r)+result

  return result

print (decimal_to_binary(32))
```

```
100000
```

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 [2]:
```
# function cubesome will take num and return sum of cube of each digit
of num
def cubesum(num):
  res = 0
  while num > 0:
    res += (int(num % 10) ** 3)
    num = int(num / 10)
  return res

# function printAmstrong will print Amstrong number from 1 to given ra
nge n
def print_amstrong(n):
  for i in range(1, n+1):
    if i == cubesum(i):
      print (i)

# function isAmstrong will return True if given num is Amstrong
def is_amstrong(num):
  return num == cubesum(num)

check_ams = 15
range_ams = 10000
print ("If given number {0} is Amstrong?".format(check_ams), is_amstro
ng(check_ams))
print ("Amstrong number from 1 to {0} are:".format(range_ams))
print_amstrong(range_ams)
```

```
If given number 15 is Amstrong? False
Amstrong number from 1 to 10000 are:
1
153
370
371
407
```

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

In [3]:
```
# function takes num as input and return product of its digits
def prod_digits(num):
  res = 1
  while num > 0:
    dig = int(num % 10)
    num = int(num / 10)
    if dig == 0:
      return 0
    res = res * dig
  return res

print (prod_digits(4134))
print (prod_digits(1240))
```

```
48
0
```

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 [36]:

```
# function returns MDR (multiplicative digital root) of n.
# 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 d
igital root of n
def MDR(num):
  num = abs(num)
  while num >= 10:
    num = prodDigits(num)
  return num

# function returns multiplicative persistance of n
# The number of times digits need to be multiplied to reach one digit
def MPersistence(num):
  count = 0
  num = abs(num)
  while num >= 10:
    num = prodDigits(num)
    count += 1
  return count

num = 1234
print ("Multiplicative Digital Root (MDR) of {0} is ".format(num), MDR
(num))
print ("Multiplicative Persistance of {0} is ".format(num), MPersisten
ce(num))
```

```
Multiplicative Digital Root (MDR) of 1234 is  8
Multiplicative Persistance of 1234 is  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, 12, 18

In [39]:
```
# function returns the sum of the proper divisor of given number num
def sumPdivisor(num):
  sum = 0
  for i in range(1, num//2 + 1):
    if num % i == 0:
      sum += i
  return sum

num = 36
print ("sum of the proper divisor of number {0} is".format(num), sumPd
ivisor(num))
```

```
sum of the proper divisor of number 36 is 55
```

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 [5]:
```
# function tells if num == sum of its proper divisors
def is_perfect(num):
  sum = 0
  for i in range(1, num // 2 + 1):
    if num % i == 0:
      sum += i
  return sum == num

# function prints perfect numbers in a range
def print_perfect(low, high):
  for i in range(low, high + 1):
    if is_perfect(i):
      print (i)

print_perfect(1, 27)
```

```
6
```

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 [6]:
```
divisors_sum = {}

# function returns the sum of the proper divisor of given number num
def sumPdivisor(num):
  if num in divisors_sum:
    return divisors_sum[num]

  sum = 0
  for i in range(1, num // 2 + 1):
    if num % i == 0:
      sum += i
  divisors_sum[num] = sum

  return sum

# function print all Amicable numbers in a range
def print_amicable(low, high):
  for i in range(low, high + 1):
    num1 = i
    num2 = sumPdivisor(i)
    if num2 < low or num2 > high or num1 >= num2:
      continue
    num3 = sumPdivisor(num2)

    if num1 == num3:
      print ("({0}, {1})".format(num1, num2))

print_amicable(1, 1590)
```

```
(220, 284)
(1184, 1210)
```

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

In [7]:
```
# function to filter only odd numbers in a list
def filter_odd(lst):
  lst1 = list(filter(lambda x: (x % 2 != 0), lst))
  return lst1

filter_odd([1, 2, 3, 4, 5, 6, 7, 8 ,9, 10, 2121])
```

Out[7]:
```
[1, 3, 5, 7, 9, 2121]
```

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

In [48]:
```
# function to map the given list to the cube of its elements
def map_cube(lst):
  cub = list(map(lambda x : x**3, lst))
  return cub

lst = [1, 2, 3, 4, 5, 6, 7]
map_cube(lst)
```

Out[48]: [1, 8, 27, 64, 125, 216, 343]

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 [8]:
```
# function return cube of even numbers filtered from the list
def filter_odd_mapCube(lst):
  cube_even_lst = list(map(lambda x: x**3, filter(lambda x : x % 2 ==
0, lst)))
  return cube_even_lst

lst = [1, 2, 3, 4, 5, 6, 7, 8, 9]
filter_odd_mapCube(lst)
```

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