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

In [1]:
```
def mult_table(num):
  for ele in range(1,11):
    print(num, '*', ele , '=', num * ele)

mult_table(int(input("Enter a number to get the multiplication table: ")))
```

```
Enter a number to get the multiplication table: 9
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
```

In [3]:
```
#using Numpy
import numpy as np

def mult_table(num):
  for ele in np.arange(1,11):
    print(num, '*', ele , '=', num * ele)

mult_table(int(input("Enter a number to get the multiplication table: ")))
```

```
Enter a number to get the multiplication table: 7
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

**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 [7]:
```
import numpy as np
range_num = 1000
twin_prime_activate = False
twin_prime_count = 0
Prime_Number_List = []
for ele in np.arange(2,range_num):
  #print('ele',ele)
  num = 2
  while (num <= ele):

    #print(num)

    if (ele % num == 0 and ele != num):
      Prime_Number = False
      break
    else:
      Prime_Number = True


    num += 1

  if (Prime_Number == True):
    #print('Prime', ele)
    Prime_Number_List.append(ele)

    if (twin_prime_count == 2):
      print('Twin Prime', First_Twin_Num, ele)
      First_Twin_Num = ''
      twin_prime_count = 1
    else:
      twin_prime_count = 1
      #print('Prime COunt = 1')

    First_Twin_Num = ele
  else:
    #print('not a Prime', ele)
    twin_prime_count = twin_prime_count + 1
    #print('Prime',twin_prime_count)
```

```
Twin Prime 3 5
Twin Prime 5 7
Twin Prime 11 13
Twin Prime 17 19
Twin Prime 29 31
Twin Prime 41 43
Twin Prime 59 61
Twin Prime 71 73
Twin Prime 101 103
Twin Prime 107 109
Twin Prime 137 139
Twin Prime 149 151
Twin Prime 179 181
Twin Prime 191 193
Twin Prime 197 199
```

```
Twin Prime 227 229
Twin Prime 239 241
Twin Prime 269 271
Twin Prime 281 283
Twin Prime 311 313
Twin Prime 347 349
Twin Prime 419 421
Twin Prime 431 433
Twin Prime 461 463
Twin Prime 521 523
Twin Prime 569 571
Twin Prime 599 601
Twin Prime 617 619
Twin Prime 641 643
Twin Prime 659 661
Twin Prime 809 811
Twin Prime 821 823
Twin Prime 827 829
Twin Prime 857 859
Twin Prime 881 883
```

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

In [5]:
```
Input_Num = int(input('Enter the Number to fetch Prime Factors '))
#print(Input_Num)
Prime_Factor_List = []
ele = 0

while(Input_Num > Prime_Number_List[0]):

  if (Input_Num % Prime_Number_List[ele] == 0):
    Prime_Factor_List.append(Prime_Number_List[ele])
    #print(Input_Num / Prime_Number_List[ele])
    Input_Num = Input_Num / Prime_Number_List[ele]
  else:
    ele += 1

print('Prime Factors of the Given Number are ',Prime_Factor_List)
```

```
Enter the Number to fetch Prime Factors 56
Prime Factors of the Given Number are  [2, 2, 2, 7]
```

**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 [10]:
```
def Factorial(num):
  return 1 if (num == 1) else (num * Factorial(num -1))

#num = 5
#print ("Factorial of {0} is {1}".format(num,Factorial(num)))
n = int(input("Enter number of Objects (value of n)"))
r = int(input("Enter number of time (value of r)"))

print("Number of permutations of {0} objects taken {1} at a time p(n,r) is {2}".f
print("Number of combinations of {0} objects taken {1} at a time c(n,r) is {2}".f
```

```
Enter number of Objects (value of n)5
Enter number of time (value of r)3
Number of permutations of 5 objects taken 3 at a time p(n,r) is 60.0
Number of combinations of 5 objects taken 3 at a time c(n,r) is 10.0
```

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

In [11]:
```
Input_Num = int(input("Enter a decimal Number "))
Binary_Num = ''
Continue_Flag = True

while(Continue_Flag == True):
  Binary_Num = Binary_Num + str(Input_Num % 2)
  Input_Num = int(Input_Num / 2)
  if (Input_Num == 0):
    Continue_Flag = False
  #print(Binary_Num)

print("Binary equivalent of the given number is
",Binary_Num [::-1])
```

```
Enter a decimal Number 50
Binary equivalent of the given number is  110010
```

**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.**

**6.1 function cubesum() that accepts an integer and returns the sum of the cubes of individual digits of that number.**

In [16]:
```
def cubesum(num):
  Result = 0
  Continue_Flag = True
  while(Continue_Flag == True):
    Result = Result + (num % 10) ** 3

    num = int( num / 10 )

    if num == 0:
      Continue_Flag = False
  #print ('Final', Result)
  return Result


print ('Sum of Cube of all digits: ', cubesum(int(input("Enter a number "))))
```

```
Enter a number 37
Sum of Cube of all digits:  370
```

**6.2 PrintArmstrong function to print all Armstrong numbers upto given max range**

In [21]:
```
#import numpy as np

def PrintArmstrong(num):

  for ele in range(1,num):
    if (isArmstrong(ele)):
      print (ele)


PrintArmstrong(int(input('Enter the Max range number: ')))
```

```
Enter the Max range number: 1000
1
153
370
371
407
```

**6.3 isArmstrong function to find out whether the given number is Prime or not**

In [23]:
```
def isArmstrong(num):
  return True if (num == cubesum(num)) else False

Input_num = int(input('Enter number to check Armstrong number or not '))

print(isArmstrong(Input_num))
```

```
Enter number to check Armstrong number or not 53
False
```

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

In [27]:
```
def proDigits(num):
  Result = 1
  Continue_Flag = True
  while(Continue_Flag == True):
    Result = Result * (num % 10)

    num = int( num / 10 )

    if num == 0:
      Continue_Flag = False
  #print ('Final', Result)
  return Result


print ('Product of all digits: ', proDigits(int(input("Enter a number: "))))
```

```
Enter a number: 342
Product of all digits:  24
```

**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 and multiplicative persistence respectively**

In [30]:
```
def MDR(num):
  Count = 0
  while(num >= 10):
    num = proDigits(num)
    #Count += 1
  return num

print('multiplicative digital root of this number is: ',MDR(int(input('Enter a nu
```

```
Enter a number to find multiplicative digital root 341
multiplicative digital root of this number is:  2
```

In [31]:
```
def MPersistence(num):
  Count = 0
  while(num >= 10):
    num = proDigits(num)
    Count += 1
  return Count

print('multiplicative persistence of this number is: ',MPersistence(int(input('En
```

```
Enter a number to find multiplicative persistence 341
multiplicative persistence of this number is:  2
```

In [33]:
```
#Combining both Functions into one

def MDR_MPersistence(num):
  Count = 0
  while(num >= 10):
    num = proDigits(num)
    Count += 1
  return num, Count

result = MDR_MPersistence(int(input('Enter a number to find multiplicative persis
print('multiplicative digital root of this number is: ',result[0])
print('multiplicative persistence of this number is: ', result[1])
```

```
Enter a number to find multiplicative persistence 341
multiplicative digital root of this number is:  2
multiplicative persistence of this number is:  2
```

**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 [36]:
```
def sumPdivisors(num):
  Result = 0
  for ele in range(1,num):
    if (num % ele == 0):
      #print('Proper Divisor ', ele)
      Result = Result + ele

  return Result

print('Sum of all Proper Divisors is: ',sumPdivisors(int(input('Enter a number to
```

```
Enter a number to get the sum of all proper divisors 36
Sum of all Proper Divisors is:  55
```

**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 [37]:
```
import numpy as np

def perfectNumbers(range):
  for ele in np.arange(range):
    if ele == sumPdivisors(ele):
      print (ele)


perfectNumbers(int(input('Enter the Max range number: ')))
```

```
Enter the Max range number: 2000
0
6
28
496
```

**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 [38]:
```
def amicable(range):
  for ele in np.arange(range):
    Temp1 = sumPdivisors(ele)
    Temp2 = sumPdivisors(Temp1)
    if (ele == Temp2) and (ele != Temp1):
      print (ele, Temp1)

amicable(int(input('Enter a Range for amicable numbers ')))
```

```
Enter a Range for amicable numbers 2000
220 284
284 220
1184 1210
1210 1184
```

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

In [40]:
```
def even_number(num):
  if num % 2 == 0:
     return num
```

In [42]:
```
Test_List = range(1,100)
print(list(filter(even_number,Test_List)))
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 4
2, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80,
82, 84, 86, 88, 90, 92, 94, 96, 98]
```

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

In [43]:
```
def cube_func(num):
  return num ** 3
```

In [46]:
```
Test_List = range(1,100)
print("Cube of all numbers")
print(list(map(cube_func,Test_List)))
```

```
Cube of all numbers
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744, 3375, 409
6, 4913, 5832, 6859, 8000, 9261, 10648, 12167, 13824, 15625, 17576, 19683, 2195
2, 24389, 27000, 29791, 32768, 35937, 39304, 42875, 46656, 50653, 54872, 59319,
64000, 68921, 74088, 79507, 85184, 91125, 97336, 103823, 110592, 117649, 12500
0, 132651, 140608, 148877, 157464, 166375, 175616, 185193, 195112, 205379, 2160
00, 226981, 238328, 250047, 262144, 274625, 287496, 300763, 314432, 328509, 343
000, 357911, 373248, 389017, 405224, 421875, 438976, 456533, 474552, 493039, 51
2000, 531441, 551368, 571787, 592704, 614125, 636056, 658503, 681472, 704969, 7
29000, 753571, 778688, 804357, 830584, 857375, 884736, 912673, 941192, 970299]
```

**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 [45]:
```
Test_List = range(1,100)
Even_List = list(filter(even_number,Test_List))
Cube_List = list(map(cube_func,Even_List))
print("Cube of Even Numbers")
print (Cube_List)
```

```
Cube of Even Numbers
[8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832, 8000, 10648, 13824, 17576, 2195
2, 27000, 32768, 39304, 46656, 54872, 64000, 74088, 85184, 97336, 110592, 12500
0, 140608, 157464, 175616, 195112, 216000, 238328, 262144, 287496, 314432, 3430
00, 373248, 405224, 438976, 474552, 512000, 551368, 592704, 636056, 681472, 729
000, 778688, 830584, 884736, 941192]
```