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

In [1]:

```
def mul(a):
    for i in range(11):
        answer = a*i
        print(str(a) + " x " + str(i) + " = " + str(answer))
```

In [2]:

```
num = int(input("Enter a number: "))
print("The multiplication table of {} is as follows: ".format(num))
mul(num)
```

```
Enter a number: 45
The multiplication table of 45 is as follows:
45 x 0 = 0
45 x 1 = 45
45 x 2 = 90
45 x 3 = 135
45 x 4 = 180
45 x 5 = 225
45 x 6 = 270
45 x 7 = 315
45 x 8 = 360
45 x 9 = 405
45 x 10 = 450
```

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 [3]:

```
# Calculate prime numbers between 1 and 1000
lst = []
for num in range(1,1000):
    if num > 1:
        isDivisible = False;
        for index in range(2,num):
            if num % index == 0:
                isDivisible = True
        if not isDivisible:
            lst.append(num)
print(lst)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 7
1, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 1
51, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499,
503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601,
607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691,
701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907,
911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
```

In [4]:

```
# Twin primes are those numbers which have a difference of two ( 2 ) between two consec
utive prime numbers.
def twin_prime(a):
    index = 0
    tup = ()
    final_result = []
    while (index + 1) < len(a):
        if (a[index+1] - a[index]) == 2:
            tup = (a[index], a[index+1])
            final_result.append(tup)
        index+=1
    return final_result
```

In [5]:

```
print("Twin prime numbers less than 1000 are: ")
twin_prime(lst)
```

```
Twin prime numbers less than 1000 are:
```

Out[5]:

```
[(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)]
```

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

In [ ]:

```
# Not Attemped
```

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 [6]:

```
def factorial_formula(x):
    product = 1
    for i in range(1,x+1):
        product = product * i
    return product

n = int(input("Enter the total number of objects n: "))
r = int(input("Enter the total number of objects r taken at a time: "))

if n < 0 or r < 0:
    print("Please enter a positive number")
elif n >= r:
    Permutation_p = int(factorial_formula(n) / factorial_formula(n-r))
    Combination_c = int(Permutation_p / factorial_formula(r))
    print("\nThe number of Permutations are {2} and the number of Combinations are {3}
 for a total of {0} objects taken {1} at a time.".format(n,r,Permutation_p,Combination_
c))
else:
    print("\nThe number of objects taken at a time cannot be more than the total number
of objects")
```

```
Enter the total number of objects n: 6
Enter the total number of objects r taken at a time: 4

The number of Permutations are 360 and the number of Combinations are 15 f
or a total of 6 objects taken 4 at a time.
```

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

In [7]:

```
def dec_to_bin(num):
    l1 = []
    while num > 0:
        b = num % 2; num = num // 2
        l1.append(b)
    l1.reverse()
    return l1
```

In [8]:

```
# Code only written for a positive number to be converted into binary
try:
    a = int(input("Enter a positive number: "))
    if a <= 0:
        raise ValueError("Error: Entered a negative number")
    else:
        print("The binary equivalent of {} is {}".format(a,dec_to_bin(a)))
except ValueError as e:
    print(e)
```

```
Enter a positive number: 29
The binary equivalent of 29 is [1, 1, 1, 0, 1]
```

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.

In [9]:

```
def cubesum(a):
    b = str(a)
    i = 0
    sum_cubes = 0
    while i < len(b):
        x = int(b[i])**3
        sum_cubes = sum_cubes + x
        i+=1
    return sum_cubes
```

In [10]:

```
try:
    num = int(input("Enter a positive integer: "))
    if num <= 0:
        raise ValueError("Error: Entered negative integer")
    else:
        print("The sum of cubes of individual digits of {0} is {1}".format(num,cubesum(
num)))
except ValueError as e:
    print(e)
```

```
Enter a positive integer: 345
The sum of cubes of individual digits of 345 is 216
```

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

In [11]:

```
def prodDigits(a):
    x = abs(a)
    b = str(x)
    i = 0
    product = 1
    while i < len(b):
        product *= int(b[i])
        i+=1
    print("The product of the digits of {0} is {1}".format(x,product))
```

In [12]:

```
num = int(input("Enter a number:"))
prodDigits(num)
```

```
Enter a number:86
The product of the digits of 86 is 48
```

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 [13]:

```
def prodDigits(a):
    b = str(a)
    i = 0
    product = 1
    while i < len(b):
        product *= int(b[i])
        i+=1
    return product
```

In [14]:

```
def MDR(b):
    while len(str(b)) > 1:
        b = prodDigits(b)
    print("The Multiplicative Digital Root of {0} is {1}".format(num,b))
```

In [15]:

```
# This function is not needed and can be merged with MDR() to get both the results and
 save the computation time
def MPersistence(c):
    count = 0
    while len(str(c)) > 1:
        c = prodDigits(c)
        count += 1
    print("The Multiplicative Persistence of {0} is {1}".format(num,count))
```

In [16]:

```
num = int(input("Enter a positive number:"))
MDR(num)
MPersistence(num)
```

```
Enter a positive number:86
The Multiplicative Digital Root of 86 is 6
The Multiplicative Persistence of 86 is 3
```

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 [17]:

```
def sumPdivisors(num):
    num1 = abs(num)
    lst = []
    i = 1;
    while i < num1:
        if num1 % i == 0:
            lst.append(i)
        i +=1;
    print("The proper divisors of {0} are {1} and the sum of these divisors is {2}" .fo
rmat(num1,lst,sum(lst)))

a = int(input("Enter a number: "))
sumPdivisors(a)
```

```
Enter a number: 36
The proper divisors of 36 are [1, 2, 3, 4, 6, 9, 12, 18] and the sum of th
ese 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 [18]:

```
num1 = int(input("Enter the starting number for a range: "))
num2 = int(input("Enter the ending number for a range: "))

lst2 = []
for num in range(num1, num2 + 1):
    i = 1
    lst1 = []
    for var in range(i, num):
        i < num
        if num % i == 0:
            lst1.append(i)
        i +=1
    sum_lst = (sum(lst1))
    if sum_lst == num:
        lst2.append(num)
print(lst2)
```

```
Enter the starting number for a range: 0
Enter the ending number for a range: 100
[0, 6, 28]
```

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 [19]:

```
def sumPdivisors(num):
    num1 = abs(num)
    lst = []
    i = 1;
    while i < num1:
        if num1 % i == 0:
            lst.append(i)
        i +=1;
    return sum(lst)
```

In [20]:

```
def amicable_numbers(num1,num2):
    lst = []
    for i in range(num1, num2 +1):
        result = sumPdivisors(i)
        if sumPdivisors(result) == i:
            tup = (i,result)
            lst.append(tup)
        i += 1
    print("\nThe pair of amicable numbers are: {}" .format(lst))
```

In [21]:

```
a = int(input("Enter the starting number for a range: "))
b = int(input("Enter the ending number for a range: "))
amicable_numbers(a,b)
```

```
Enter the starting number for a range: 200
Enter the ending number for a range: 300

The pair of amicable numbers are: [(220, 284), (284, 220)]
```

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

In [22]:

```
a_lst = [1,2,3,4,5,6,7,8,9,10,11]

odd_lst = list(filter(lambda x: (x % 2 != 0), a_lst))
print(odd_lst)
```

```
[1, 3, 5, 7, 9, 11]
```

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

In [23]:

```
numbers = [1,2,3,4,5,6,7,8,9,10]

def powerOfThree(num):
    return num**3

cubed = list(map(powerOfThree, numbers))
print(cubed)
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

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 [24]:

```
numbers = [1,2,3,4,5,6,7,8,9,10]

def powerOfThree(num):
    return num**3

even_lst = list(filter(lambda x: (x % 2 == 0), numbers))

even_cubed = list(map(powerOfThree, even_lst))
print(even_cubed)
```

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