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

In [1]:

```
#function to print multiplication table of a number
def multiply(num):
    for i in range(1,11):
        print(num,"*",i,"=",num*i)
    return
multiply(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
```

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 [2]:

```python
#Program to print twin primes less than 1000
import math
def isPrime(num):
    for i in range(2,int(math.sqrt(num))+1):
        if num%i==0:
            return False
    return True
def twinPrime():
    for i in range(3,1000,2):
        if isPrime(i)==isPrime(i+2)==True:
            print("(",i,",",i+2,")")
    return
twinPrime()
```

```
( 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 [3]:
```

```
#Program to find prime factors
def primeFact(num):
    while num%2==0:
        print(2)
        num=num/2
    for i in range(3,int(num)+1,2):
        while num%i==0:
            print(i)
            num=num/i
primeFact(63)
```

```
3
3
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 [4]:
```

```
#Program to implement formulae of permutations and combinations
def fact(n):
    if n==1:
        return n
    return n*fact(n-1)
def permutations(n,r):
    return int(fact(n)/fact(n-r))
def combinations(n,r):
    return int(permutations(n,r)/fact(r))
print("No of Permutations: ",permutations(6,4))
print("No of Combinations: ",combinations(6,4))
```

```
No of Permutations:  360
No of Combinations:  15
```

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

```
In [5]:
```

```
#Function to convert decimal number to binary number
def binConversion(n):
    i=1
    res=0
    while n>0:
        res=res+(n%2)*i
        i=i*10
        n=int(n/2)
    return res
print("Binary format of given number:",binConversion(11))
```

```
Binary format of given number: 1011
```

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 [6]:

```
#Functions to find a sum of cube of digits of a number, to check a number is Armstrong
 or not and to print it
def cubeSum(n):
    res=0
    while n>0:
        res=res+((n%10)**3)
        n=int(n/10)
    return res
def isArmStrong(num):
    return (num==cubeSum(num))
def printArmStrong(num):
    if isArmStrong(num):
        print(num,"is an Armstrong number.")
    else:
        print(num,"is not an Armstrong number.")
printArmStrong(153)
```

```
153 is an Armstrong number.
```

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

In [7]:

```
#Function to return the product of digits of a number
def prodDigits(num):
    if num==0:
        return 1
    return (num%10)*prodDigits(int(num/10))
print("Product of Digits:",prodDigits(189))
```

```
Product of Digits: 72
```

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 [8]:

```
# Functions to return multiplicative digital root and multiplicative persistence of giv
en number
def MDR(num):
    if int(num/10)==0:
        return num
    return MDR(prodDigits(num))
def MPersistence(num):
    count=0
    while int(num/10)!=0:
        num=prodDigits(num)
        count=count+1
    return count
print("MDR:",MDR(341))
print("MPersistence:",MPersistence(341))
```

```
MDR: 2
MPersistence: 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 [9]:

```
#Function to find sum of proper divisors of number
import math
def sumPdivisors(num):
    if num==1:
        return 0
    sum=1
    for i in range(2,int(math.sqrt(num))+1):
        if (num%i)==0:
            sum=sum+i+int(num/i)
    return sum
print("Sum of proper divisors:",sumPdivisors(6))
```

```
Sum of proper divisors: 6
```

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 [10]:

```
#Program to print perfect numbers in given range
def printPerfectNumber(start,end):
    print("Perfect numbers between",start,"and",end)
    for i in range(start,end+1):
        if i==sumPdivisors(i):
            print(i)
    return
printPerfectNumber(1,1000)
```

```
Perfect numbers between 1 and 1000
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 [11]:

```
#Function to print pairs of amicable numbers in a range
def printAmicableNumber(start,end):
    print("Amicable numbers between",start,"and",end)
    list1=[]
    for i in range(start,end+1):
        if not(i in list1):
            res=sumPdivisors(i)
            if res<=end and i!=res and res!=0:
                if i==sumPdivisors(res):
                    list1.append(i)
                    list1.append(res)
                    print("(",i,",",res,")")
printAmicableNumber(1,7000)
```

```
Amicable numbers between 1 and 7000
( 220 , 284 )
( 1184 , 1210 )
( 2620 , 2924 )
( 5020 , 5564 )
( 6232 , 6368 )
```

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

In [12]:

```
#Program to filter odd numbers from give list
def oddList(input):
    return list(filter(lambda x:x%2!=0,input))
print("List of Odd Numbers:",oddList([1,2,3,4,5,6]))
```

```
List of Odd Numbers: [1, 3, 5]
```

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

In [13]:

```
#Program to apply map() on elements of given list
def cube(input):
    return list(map(lambda x:x**3,input))
print("Cube of elements in List:",cube([1,2,3,4]))
```

```
Cube of elements in List: [1, 8, 27, 64]
```

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 [14]:

```
#Program to apply map() and filter() on elements of given list
def cubeOfEven(input):
    return list(map(lambda x:x**3,filter(lambda x:x%2==0,input)))
print("Cube of Even numbers in list",cubeOfEven([1,4,6,8,9,20,3]))
```

```
Cube of Even numbers in list [64, 216, 512, 8000]
```

In [ ]:

```

```