Lab1

Write an algorithm and a program to check one integer is a divisor of the other if two integers are given

1. Input two integers from the user
2. Check which is the larger number among the both
3. If the first number is larger than the second number, divide the first number by the second number and check the remainder If the second number is zero, then the division is possible
4. If the second number is larger than the first number, divide the second number by the first number and check the remainder. If the first number is zero then the division is not possible
5. If the remainder obtained is zero then one integer is the divisor of the other , otherwise not.
6. If both the numbers are same, then the remainder obtained is zero then we say that one number is a divisor of itself. If any one number is kept zero, then both the numbers are zero, and inderterminant form is obtained so that division cannot takes place
7. If any one of the number is zero, then there can be two possibilities that the other number can divide zero but zero cannot divide the number

In [3]:

```python
num1 = int(input("Enter the first integer: "))
num2 = int(input("Enter the second integer: "))

if num1 % num2 == 0:
    print(f"{num2} is a divisor of {num1}.")
elif num2 % num1 == 0:
    print(f"{num1} is a divisor of {num2}.")
else:
    print("Neither number is a divisor of the other.")

```

```
Enter the first integer: 22
Enter the second integer: 11
11 is a divisor of 22.
```

Write the algorithm and prime to find all the prime divisors of a number

Algorithm to Find All Prime Divisors of a Number Start: Begin the process by preparing to find the prime divisors of a number.

Input the Number: Ask the user to enter a positive integer.

Check for Special Cases:

If the number entered is less than or equal to 1: Print "No prime divisors" because numbers less than or equal to 1 do not have any prime divisors. End the process. Prepare to Find Prime Divisors:

Create an empty list named prime_divisors to store the prime divisors of the input number. Loop to Check Divisors:

Start with the smallest possible divisor, which is 2. Check all numbers starting from 2 and increasing one by one, up to the square root of the input number. For each number in this range: Check if it divides the input number without leaving a remainder (i.e., it is a divisor). If it is a divisor: Add the number to the prime_divisors list. Divide the input number by this divisor repeatedly until it can no longer be divided evenly. This step ensures that all multiples of the divisor are removed from the input number. Check for Any Remaining Prime Factor:

After completing the loop, check if the input number is still greater than 1. If it is, this remaining number is a prime factor. Add it to the prime_divisors list. Output the Result:

Print the list of prime divisors, which contains all the prime numbers that divide the original input number. End:

In [6]:

```
import math

def find_prime_divisors(n):
    if n <= 1:
        return "No prime divisors"

    prime_divisors = []
    # Check divisors from 2 to sqrt(n)
    for i in range(2, int(math.sqrt(n)) + 1):
        # If i divides n, it is a divisor
        if n % i == 0:
            # Check if i is prime (it will be if it divides n completel
            prime_divisors.append(i)
            # Remove all occurrences of i from n
            while n % i == 0:
                n //= i

    # If n is still greater than 1, it is a prime number
    if n > 1:
        prime_divisors.append(n)

    return prime_divisors

# Input from user
num = int(input("Enter an integer: "))
result = find_prime_divisors(num)

if isinstance(result, list):
    print(f"The prime divisors of {num} are: {result}")
else:
    print(result)

```

```
Enter an integer: 675
The prime divisors of 675 are: [3, 5]
```

Write an algorithm and a program to list out all the integers which are less than or equal to and relatively prime to that integer

1. Input an integer from user

2. Consider another integer and check if these two integers are relatively prime or not
3. If the gcd of those two integers is 1 then these are said to be relatively prime
4. Those integers whose gcd with num is 1 is stored in a list and that list contains all those integrs which are relatively prime to that integer and less than it

In [17]:

```
import math

def find_relatively_prime_numbers(n):

    relatively_prime_numbers = []


    for i in range(1, n + 1):

        if math.gcd(i, n) == 1:
            relatively_prime_numbers.append(i)

    return relatively_prime_numbers


num = int(input("Enter a positive integer: "))


if num <= 0:
    print("Please enter a positive integer.")
else:

    result = find_relatively_prime_numbers(num)

    print(f"Integers less than or equal to {num} that are relatively pr
    print(result)

```

```
Enter a positive integer: 23
Integers less than or equal to 23 that are relatively prime to 23:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
1, 22]
```

Write an algorithm and a program to determine the number of non-negative divisors and the sum of non-negative divisors of a given integers

1. Input a number from the user
2. If the number is zero it has all integers as divisors except 0 since the division of 0 by 0 is an indeterminate form
3. If the number is a prime number number then only two divisor exists 1 and the number itself
4. If the number is a composite number, then all the numbers which divide the given number will have remainder 0
5. Consider a variable count and assign its value to 0. On executing the for loop each time and if a divisor is obtained then increment the value of count by 1
6. Consider anoter variable sum and assign its value to 0. On executing the for loop each time and if a divisor obtained then add the divisor to the variable sum and on the final iteration, the sum of all the non-negative divisors is obtained

In [13]:
```
def calculate_divisors(n):
    # Initialize counters
    divisor_count = 0
    divisor_sum = 0

    # Loop through all numbers from 1 to n
    for i in range(1, n + 1):
        if n % i == 0:
            divisor_count += 1
            divisor_sum += i

    return divisor_count, divisor_sum

# Input from the user
num = int(input("Enter a positive integer: "))

# Ensure the input is valid
if num <= 0:
    print("Please enter a positive integer.")
else:
    # Calculate divisors and their sum
    count, total_sum = calculate_divisors(num)
    # Display results
    print(f"The number of non-negative divisors of {num} is: {count}")
    print(f"The sum of non-negative divisors of {num} is: {total_sum}"
```

```
Enter a positive integer: 34
The number of non-negative divisors of 34 is: 4
The sum of non-negative divisors of 34 is: 54
```

In [ ]:
```

```

Write an algorithm and a program to verify the fundamental theorem of arithmetic

1. Input a number from user
2. Find out the prime factor of that number
3. Iterate the checking of prime factors in a for loop and multiply each prime factor successively in order to get the prime number factorisation of that number

In [20]:

```
def prime_factorization(n):
    prime_factors = []
    i = 2


    while i * i <= n:
        while n % i == 0:
            prime_factors.append(i)
            n //= i
        i += 1
    if n > 1:
        prime_factors.append(n)

    return prime_factors

def verify_fundamental_theorem(n):
    # Find the prime factors of n
    prime_factors = prime_factorization(n)


    product = 1
    for factor in prime_factors:
        product *= factor


    if product == n:
        return prime_factors, True
    else:
        return prime_factors, False


num = int(input("Enter an integer greater than 1: "))


if num <= 1:
    print("The fundamental theorem of arithmetic is valid only for inte
else:
    factors, is_valid = verify_fundamental_theorem(num)
    print(f"Prime factors of {num}: {factors}")
    if is_valid:
        print(f"The product of the prime factors equals {num}. The theo
    else:
        print(f"The theorem is not verified for {num}.")

```

```
Enter an integer greater than 1: 234
Prime factors of 234: [2, 3, 3, 13]
The product of the prime factors equals 234. The theorem is verified!
```

Division Algorithm

1. Take two user inputs which are integers
2. Whichever of the inputs is the greater number is assigned to variable x. This will be expressed in terms of the smaller value y
3. The division algorithm expresses the largest number in terms of the lower number. The highest multiple of the smaller number which is lower than the larger number is computed by dividing the larger number by smaller. This will not always result in an

integer. However, the quotient that is obtained by performing this operation is the same as using the floor function over x/y. This gives the value of b in division algorithm

4. Now, to obtain the remainder the modulus operator is used. This gives value of r in the division algorithm
5. It is verified that the remainder that is obtained is lesser than smaller number b

In [14]:
```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
a, b = max(a, b), min(a, b)
q = a//b
r = a%b
print(f"{a} = {b} * {q} + {r}")
```

```
Enter first number: 56
Enter second number: 23
56 = 23 * 2 + 10
```

In [ ]:
```

```

Number Systems

Q. Write an algorithm and a program to convert decimal number to binary system

1. The decimal number that is to be converted to binary is taken as an input from the user
2. An empty list is created which would store the binary equivalent of the number
3. A while loop is created which runs for as many times as is required to bring down the number to 1 by repeatedly dividing the number by 2. If it is divisible by 2 the remainder is 0 and otherwise the number is 1. The list l stores it in that order
4. The list l is printed in reverse order

Program to Convert a Decimal Number to Binary

In [15]:
```
a = int(input("Enter a number: "))
binary_list = []
while a!=0:
    binary_list.append(a%2)
    a//=2
binary_list[::-1]
```

```
Enter a number: 76
```

Out[15]:
```
[1, 0, 0, 1, 1, 0, 0]
```

Lab 2

Prime Divisors

Q. List out all prime divisors of a given integer

Take an input positive integer from user. Create a list l which is going to contain all divisors of number n. Add 1 to it as it will always be a divisor. The complexity od program can be reduced to doing so. Create a loop(a for loop) which runs for the number of iterations as there are numbers equal to the number. At every iteration the control variable increments by 1. Thus at end of the loop. we have every number from 1 to n which is taken by variable. At every step, check if the control variable divides the number n. This is done by obtaining the modulus of the number with control variable. If it results in 0, then it means that the number divides n. If the number divides n, it means it is a divisor. Now, we have to check if it is a prime. For this, another for loop is created which runs for the number of iteration equal to divisor. If the number of divisors of the divisor are 1, which is itself(since the loop starts running from 2 a we have already appended 1 to the list) then it is a prime divisor. Append it to the list.

In [9]:
```
n=int(input("Enter a positive integer:"))
l=[1]
for i in range(1,n+1):
    if n%i==0:
        l1=[]
        #this means i is a divisor
        #checking i is prime
        for j in range(2,i+1):
            if i%j==0:
                l1.append(j)
        if len(l1)==1:
            l.append(i)
print(l)
```

```
Enter a positive integer:2
[1, 2]
```

Q. Find GCD and LCM of any two numbers

In [10]:
```
from sympy import *
def prime_factorization(n):
    l=[]
    p=[]
    while n!=1:
        for i in range(2,n+1):
            if n%i==0 and isprime(i)==True: #is prime divisor
                if i in l:
                    p[l.index(i)]+=1
                else:
                    l.append(i)
                    p.append(1)
                n=int(n/i)
    return l,p

a=int(input("Enter the first number: "))
b=int(input("Enter the second number: "))
a_factors=prime_factorization(a)[0]
a_frequency=prime_factorization(a)[1]
b_factors=prime_factorization(b)[0]
b_frequency=prime_factorization(b)[1]

gcd=[]
gcdf=[]
for i in range(0,len(a_factors)):
    if a_factors[i] in b_factors:
        gcd.append(a_factors[i])
        gcdf.append(min(a_frequency[i],b_frequency[i]))
    else:
        gcd.append(a_factors[i])
        gcdf.append(a_frequency[i])
for i in range(0, len(b_factors)):
    if b_factors[i] in a_factors:
        pass
    else:
        gcd.append(b_factors[i])
        gcdf.append(b_frequency[i])

g=1
for i in range(len(gcd)):
    g*=gcd[i]**gcdf[i]
print("GCD=",g)
```

```
Enter the first number: 34
Enter the second number: 23
GCD= 782
```

Lab 3

Q. Write an algorithm and program to convert one number system to all other number systems.

1. Input a number from the user in one number system, say decimal number system

2. For converting it into binar number system, divide the number by 2 and find out its remainders by successively dividing by 2 till 1 is not obtained as the last dividend. Write the remainders obtained in the ascending form.
3. For converting it into octal number system, divide the number by 8 and find out its remainder by successively dividing by 8 till 1 is not obtained as the last dividend. Write the remainders obtained in the ascending form.
4. For converting it into hexadecimal number system, divide the number by 16 and find out its remainder by successively dividing by 16 till 1 is not obtained as the last dividend. Write the remainders obtained in the ascending form.

In [11]:
```
def decimal_to_binary(decimal_number):
    binary_result=""
    while decimal_number>0:
        remainder=decimal_number%2
        binary_result=str(remainder)+binary_result
        decimal_number=decimal_number//2
    return binary_result

def decimal_to_octal(decimal_number):
    octal_result=""
    while decimal_number>0:
        remainder=decimal_number%8
        octal_result=str(remainder)+octal_result
        decimal_number=decimal_number//8
    return octal_result

def decimal_to_hex(decimal_number):
    hex_char="0123456789ABCDEF"
    hex_result=""
    while decimal_number>0:
        remainder=decimal_number%16
        hex_result=hex_char[remainder]+hex_result
        decimal_number=decimal_number//16
    return hex_result
```

In [12]:
```
print(decimal_to_binary(100))
print(decimal_to_octal(100))
print(decimal_to_hex(100))
```

```
1100100
144
64
```

Q. Write an algorithm and a program to verify the Fundamental theorem of Arithmetic

In [13]:
```
num = int(input("Enter the number:"))
if num==0 or num==1:
    print(num,"is neither prime nor composite")
elif num>1:
    result="1*"
    l=[]
    for i in range(1,num+1):
        if num%i==0:
            l1=[]
            #this means i is a divisor
            #checking i is prime
            for j in range(2,i+1):
                if i%j==0:
                    l1.append(j)
            if len(l1)==1:
                l.append(i)
    print(l)
    while num!=1:
        for i in l:
            if num%i==0:
                num//=i
                result+=str(i)+"*"
    print(result[:-1])
```

```
Enter the number:23
[23]
1*23
```

In [ ]:
```

```

Lab 4

System of linear congruences, divisibility test of 2,3,4,5,6,7,8,9,10,11,12,13

Write an algorithm and a program to find the solution of a given system of linear congruences

1. Consider two integers a1 and b1.
2. The linear congruence a1 is congruent to b1(mod n1) has a solution if gcd(a1,n1)|b1.
3. The equation formed is a1x + n1y = b1 and using the reverse method, the values of x0 and y0 are determined.
4. Then the solutions are x0, x0 + n1/d1, x0 + 2n1/d1, x0 + 3n1/d1,...,x0 + (d1-1)n1/d1 where d1 = gcd(a1,n1).
5. Similarly, there will be different linear congruences such as a2 is congruent to b2(mod n2) and so on.

In [5]:

```python
from sympy.ntheory.modular import solve_congruence



def solve_congruences(congruences):
    return solve_congruence(*congruences)


def check_divisibility(n):
    rules = {
        2: n % 2 == 0,
        3: sum(map(int, str(n))) % 3 == 0,
        4: n % 4 == 0,
        5: n % 5 == 0,
        6: n % 6 == 0,
        7: n % 7 == 0,
        8: n % 8 == 0,
        9: sum(map(int, str(n))) % 9 == 0,
        10: n % 10 == 0,
        11: sum(int(d) if i % 2 == 0 else -int(d) for i, d in enumerate
        12: n % 12 == 0,
        13: n % 13 == 0
    }
    return {k: "Divisible" if v else "Not Divisible" for k, v in rules.


congruences = [(2, 3), (3, 5), (2, 7)]
solution = solve_congruences(congruences)
print("Solution to congruences:", solution)

n = 123456
print("Divisibility results:", check_divisibility(n))

```

```
Solution to congruences: (23, 105)
Divisibility results: {2: 'Divisible', 3: 'Divisible', 4: 'Divisible', 5:
'Not Divisible', 6: 'Divisible', 7: 'Not Divisible', 8: 'Divisible', 9: 'N
ot Divisible', 10: 'Not Divisible', 11: 'Not Divisible', 12: 'Divisible',
13: 'Not Divisible'}
```

In [6]:
```
from sympy.ntheory.modular import solve_congruence

def solve_linear_congruences(congruences):
    """
    Solve a system of linear congruences using the Chinese Remainder Th

    Parameters:
    congruences: List of tuples (a, m) where x ≡ a (mod m).

    Returns:
    The smallest non-negative solution x.
    """
    solution = solve_congruence(*congruences)
    return solution


congruences = [(2, 3), (3, 5), (2, 7)]  # Example: x ≡ 2 (mod 3), x ≡ 3
solution = solve_linear_congruences(congruences)

print("Solution to the system of congruences:", solution)

```

```
Solution to the system of congruences: (23, 105)
```

Conversions of Number Systems

In [1]:

```
def decimal_to_binary(n):
    return bin(n)[2:]

def decimal_to_octal(n):
    return oct(n)[2:]

def decimal_to_hexadecimal(n):
    return hex(n)[2:]

def binary_to_decimal(b):
    return int(b, 2)

def octal_to_decimal(o):
    return int(o, 8)

def hexadecimal_to_decimal(h):
    return int(h, 16)

def binary_to_octal(b):
    decimal = binary_to_decimal(b)
    return decimal_to_octal(decimal)

def binary_to_hexadecimal(b):
    decimal = binary_to_decimal(b)
    return decimal_to_hexadecimal(decimal)

def octal_to_binary(o):
    decimal = octal_to_decimal(o)
    return decimal_to_binary(decimal)

def octal_to_hexadecimal(o):
    decimal = octal_to_decimal(o)
    return decimal_to_hexadecimal(decimal)

def hexadecimal_to_binary(h):
    decimal = hexadecimal_to_decimal(h)
    return decimal_to_binary(decimal)

def hexadecimal_to_octal(h):
    decimal = hexadecimal_to_decimal(h)
    return decimal_to_octal(decimal)

if __name__ == "__main__":
    decimal_num = 29
    binary_num = '11101'
    octal_num = '35'
    hex_num = '1D'

    print(f"Decimal {decimal_num} to Binary: {decimal_to_binary(decima
    print(f"Decimal {decimal_num} to Octal: {decimal_to_octal(decimal_r
    print(f"Decimal {decimal_num} to Hexadecimal: {decimal_to_hexadecin

    print(f"Binary {binary_num} to Decimal: {binary_to_decimal(binary_r
    print(f"Binary {binary_num} to Octal: {binary_to_octal(binary_num)}
    print(f"Binary {binary_num} to Hexadecimal: {binary_to_hexadecimal(

    print(f"Octal {octal_num} to Decimal: {octal_to_decimal(octal_num)}
    print(f"Octal {octal_num} to Binary: {octal_to_binary(octal_num)}")
    print(f"Octal {octal_num} to Hexadecimal: {octal_to_hexadecimal(oct

    print(f"Hexadecimal {hex_num} to Decimal: {hexadecimal_to_decimal(h
```

```
    print(f"Hexadecimal {hex_num} to Binary: {hexadecimal_to_binary(hex
    print(f"Hexadecimal {hex_num} to Octal: {hexadecimal_to_octal(hex_n
```

```
Decimal 29 to Binary: 11101
Decimal 29 to Octal: 35
Decimal 29 to Hexadecimal: 1d
Binary 11101 to Decimal: 29
Binary 11101 to Octal: 35
Binary 11101 to Hexadecimal: 1d
Octal 35 to Decimal: 29
Octal 35 to Binary: 11101
Octal 35 to Hexadecimal: 1d
Hexadecimal 1D to Decimal: 29
Hexadecimal 1D to Binary: 11101
Hexadecimal 1D to Octal: 35
```

In [ ]: