

# **23MAT116 DISCRETE MATHEMATICS**

**LAB MANUAL**

**by:**

Akshay KS

CB.SC.U4CSE23104

DEPARTMENT OF MATHEMATICS

# AMRITA VISHWA VIDYAPEETHAM

COIMBATORE CAMPUS (INDIA)

|  |  |
| --- | --- |
| S.NO | TITLE |
| 1. | TRUTH TABLES |
| 2. | TAUTOLOGY, CONTINGENCY, CONTRADCTION |
| 3. | QUANTIFIERS |
| 4. | FACTORIAL OF A NUMBER |
| 5. | FINDING GCD |
| 6. | FINDING LCM |
| 7. | MODULAR EXPONENTIATION |
| 8. | POWER OF A NUMBER |
| 9. | FIBONACCI SERIES |
| 10. | GCD DIVIDES LCM |
| 11. | FLOYD’S ALGORITHM |
| 12. | WARSHALL’S ALGORITHM |
| 13. | TOWER OF HANOI |

1.Truth Tables

Code:

def conjn(p,q):

return p and q

def disjn(p,q):

return p or q

def impln(p,q):

return not p or q

def Bi\_condn(p,q):

return (not p or q) and (p or not q)

def XOR(p,q):

return (not p and q) or (p and not q)

def negation(p):

return not p

print(" Conjunction ")

print("-------------------------------------------------")

print("| p | q | r |")

print("-------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

a=conjn(p,q)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',a,'\t','|')

print("--------------------------------------------------")

print(" Disjunction ")

print("-------------------------------------------------")

print("| p | q | r |")

print("-------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

a=disjn(p,q)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',a,'\t','|')

print("--------------------------------------------------")

print(" Implication ")

print("-------------------------------------------------")

print("| p | q | r |")

print("-------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

a=impln(p,q)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',a,'\t','|')

print("--------------------------------------------------")

print(" Bi-Conditinal ")

print("-------------------------------------------------")

print("| p | q | r |")

print("-------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

a=Bi\_condn(p,q)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',a,'\t','|')

print("--------------------------------------------------")

print(" XOR ")

print("-------------------------------------------------")

print("| p | q | r |")

print("-------------------------------------------------")

for p in [True , False]:

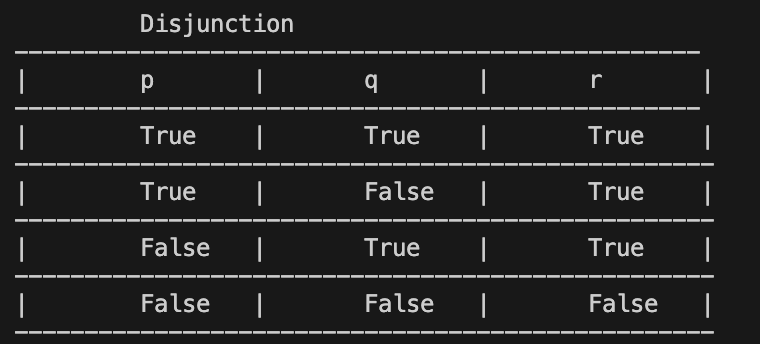
for q in [True , False]:

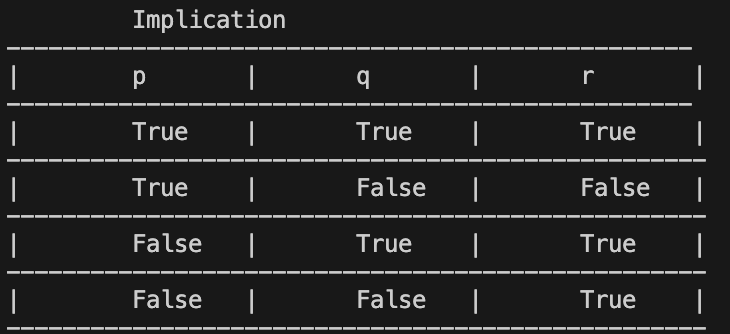
a=XOR(p,q)

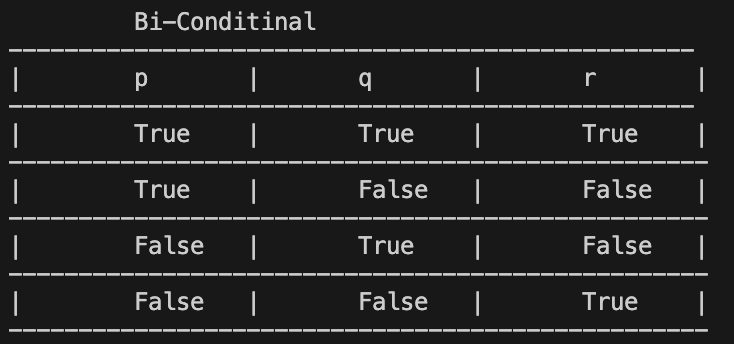
print('|','\t',p,'\t','|','\t',q,'\t','|','\t',a,'\t','|')

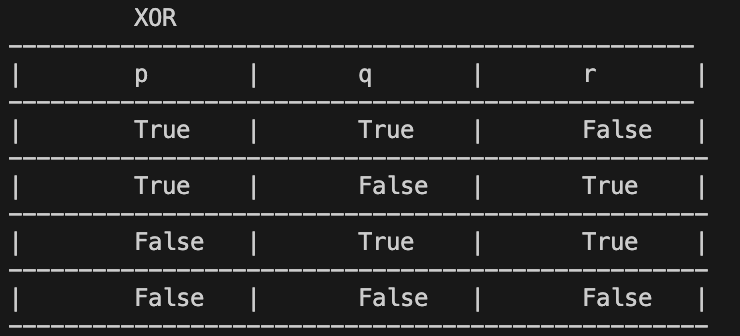
print("--------------------------------------------------")

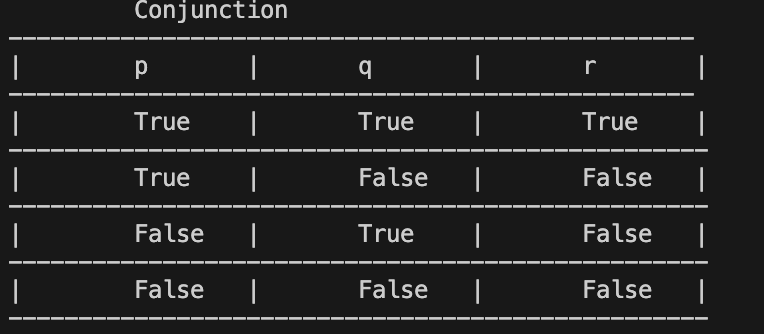
Output:











2.Python program for checking tautology, contingency and contradiction

1. (p -> q) v (q -> r)
2. (p -> q) ∧ (q -> r)
3. ((p -> q) ∧(q -> r) ) -> (p -> r)
4. ((p v q) ∧(p -> r)) -> (q v r)
5. ((p v q) ∧(p -> r) ∧ (q -> r)) -> r

Code:

def q1(p,q,r):

return (not p or q) or (not q or r)

def q2(p,q,r):

return (not p or q) and (not q or r)

def q3(p,q,r):

return (not((not p or q) and (not q or r)) or (not p or q))

def q4(p,q,r):

return(not((p or q)and(not p or r)) or (q or r))

def q5(p,q,r):

return(not((p or q) and (not p or q) and (not q or r)) or r)

def tauto(q1):

return

print(" q1 ")

print("------------------------------------------------------------------")

print("| p | q | r | a |")

print("------------------------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q1(p,q,r)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',r,'\t','|','\t',a,'\t','|')

print("------------------------------------------------------------------")

s=True

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q1(p,q,r)

s= s and a

if s == True:

print('q1 is a Tautology')

else:

print("q2 is not a Tautology")

print(" q2 ")

print("------------------------------------------------------------------")

print("| p | q | r | a |")

print("------------------------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q2(p,q,r)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',r,'\t','|','\t',a,'\t','|')

print("------------------------------------------------------------------")

s2=True

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q2(p,q,r)

s2= s2 and a

if s2 == True:

print('q2 is a Tautology')

else:

print("q2 is not a Tautology")

print(" q3 ")

print("------------------------------------------------------------------")

print("| p | q | r | a |")

print("------------------------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q3(p,q,r)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',r,'\t','|','\t',a,'\t','|')

print("------------------------------------------------------------------")

s3=True

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q3(p,q,r)

s3= s3 and a

if s3 == True:

print('q3 is a Tautology')

else:

print("q3 is not a Tautology")

print(" q4 ")

print("------------------------------------------------------------------")

print("| p | q | r | a |")

print("------------------------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q4(p,q,r)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',r,'\t','|','\t',a,'\t','|')

print("------------------------------------------------------------------")

s4=True

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q4(p,q,r)

s4= s4 and a

if s4 == True:

print('q4 is a Tautology')

else:

print("q2 is not a Tautology")

print(" q5 ")

print("------------------------------------------------------------------")

print("| p | q | r | a |")

print("------------------------------------------------------------------")

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q5(p,q,r)

print('|','\t',p,'\t','|','\t',q,'\t','|','\t',r,'\t','|','\t',a,'\t','|')

print("------------------------------------------------------------------")

s5=True

for p in [True , False]:

for q in [True , False]:

for r in [True,False]:

a=q5(p,q,r)

s5= s5 and a

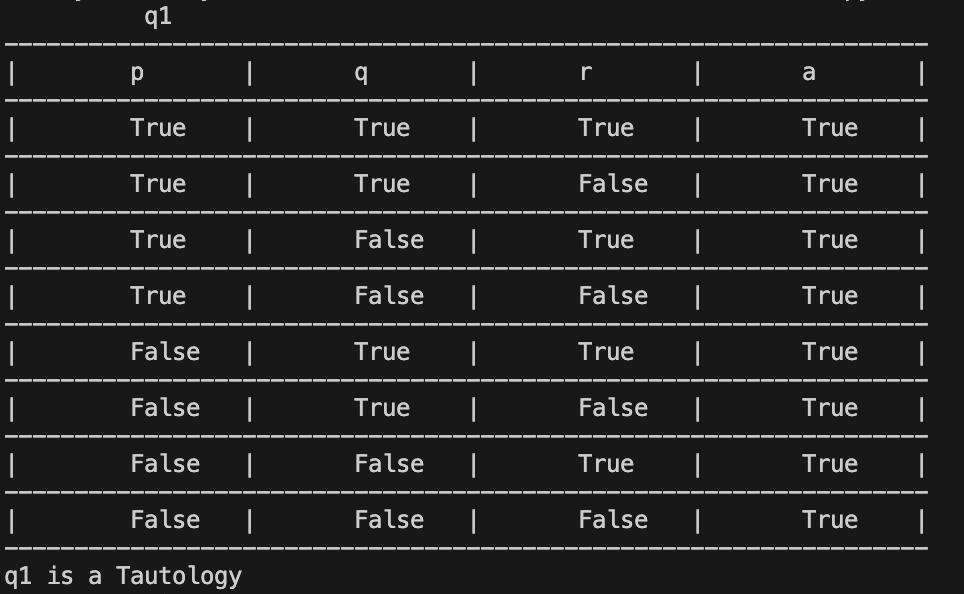
if s5 == True:

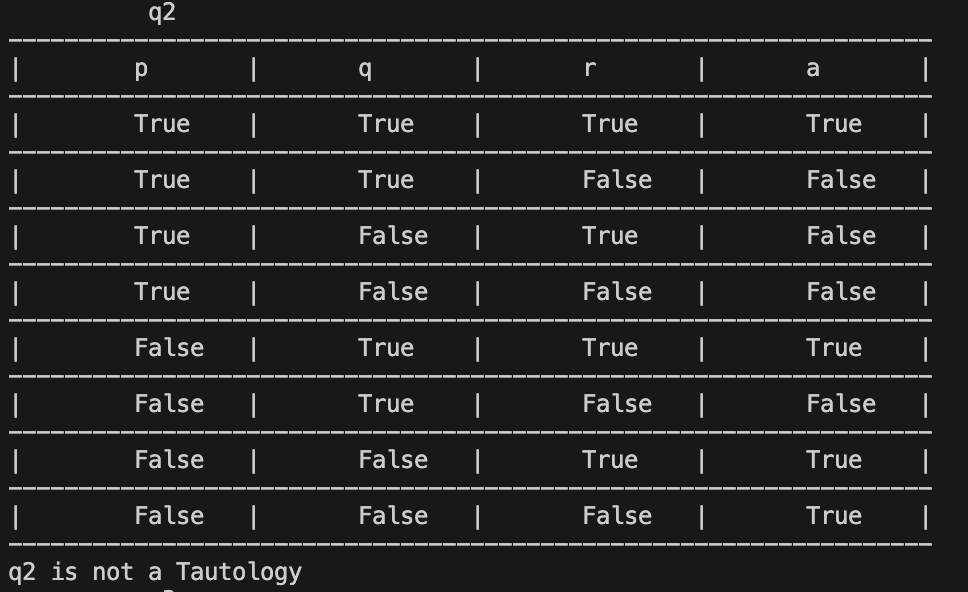
print('q5 is a Tautology')

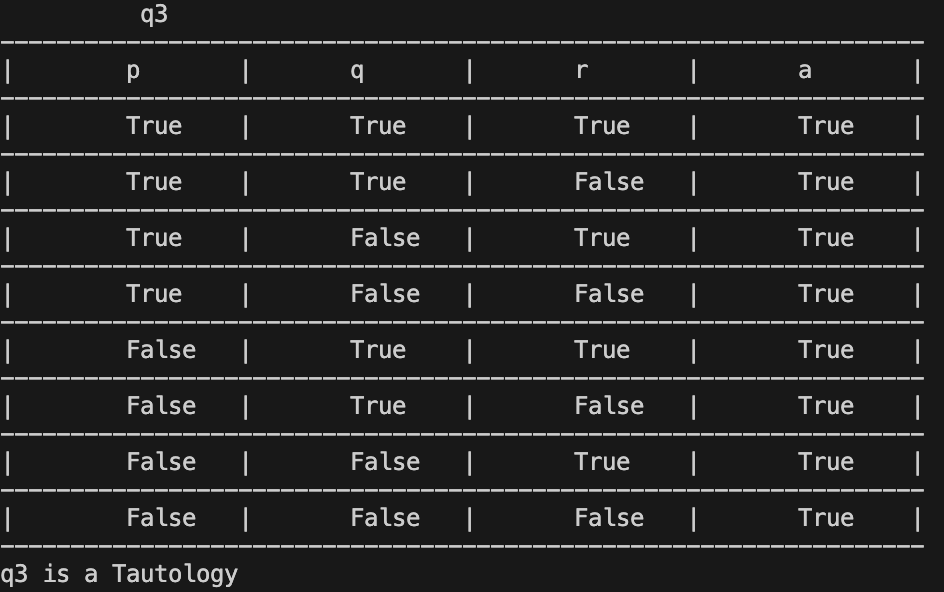
else:

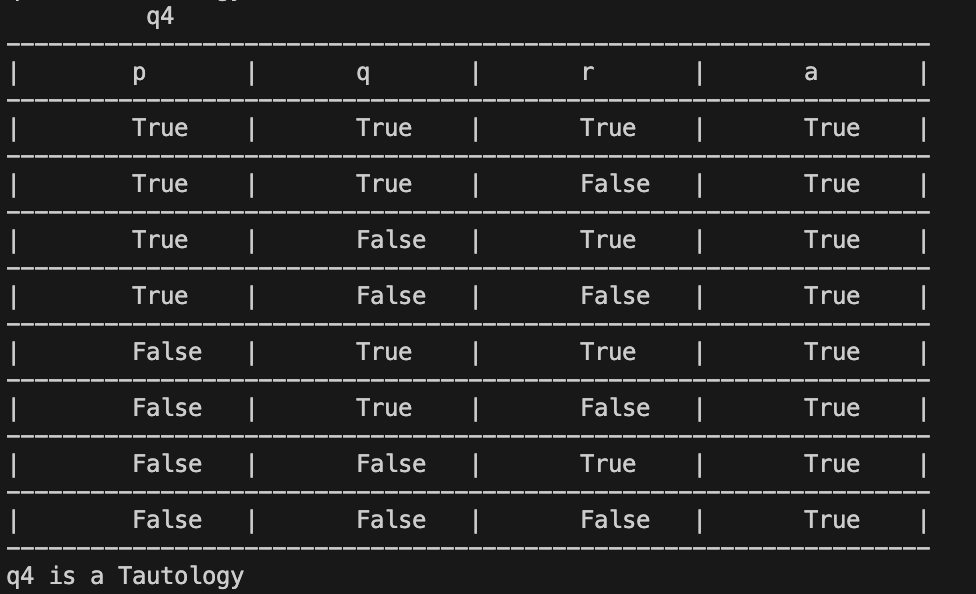
print("q5 is not a Tautology")

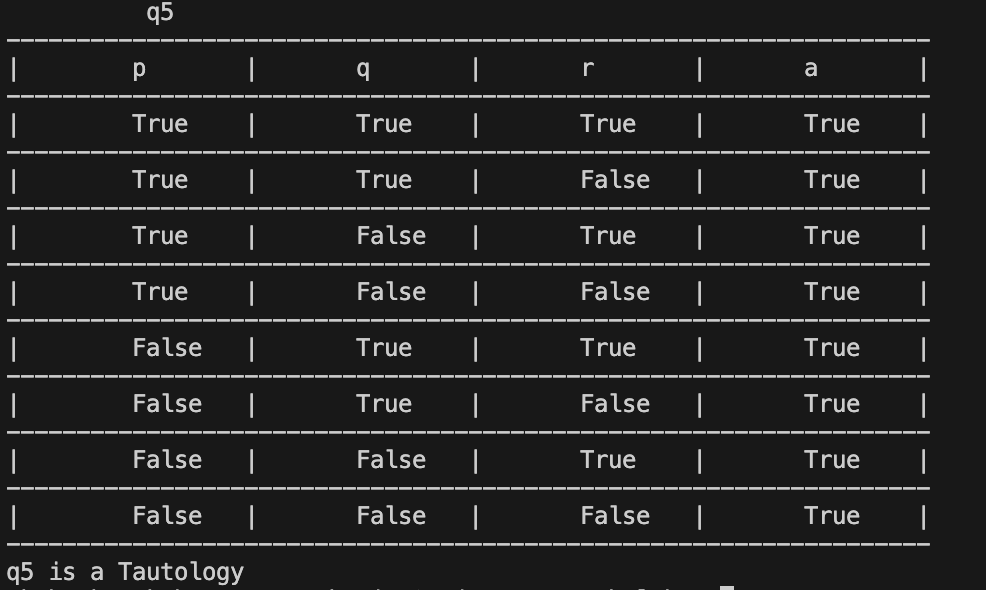
Output:











3.Quantifiers

i)Universal quantifier

Code:

stmt = "For all x , P(x)."

def P(x):

return x+1 > x

forall = True

for x in [-2,-1,0,1,2]:

# Check if at least 1 is false

if P(x) == False:

forall = False

print(stmt)

print(forall)

Output:



ii)Existential Quantifier

Code:

stmt = "There exists an x, such that P(x)."

def P(x):

return x\*\*2 == 4

exists = False

for x in [-2,-1,0,1,2]:

# Check if at least 1 is false

if P(x) == True:

exists = True

print(stmt)

print(exists)

Output:



4.Factorial

Code:

def Factorial(n):

if n==1:

return 1

else:

return n\*Factorial(n-1)

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

print(Factorial(num))

Output:



5.G.C.D of a number

Code:

def gcd(a,b):

if a==0:

return b

else:

return gcd(b%a,a)

num1 = int(input("Enter the number 1 : "))

num2 = int(input("Enter the number 2 : "))

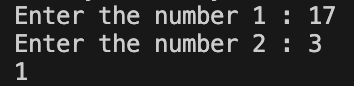
if num1>num2:

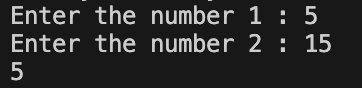
print(gcd(num2,num1))

else:

print(gcd(num1,num2))

Output:





6.LCM of two numbers

Code:

def lcm(x, y):

# selecting the greater number

if x > y:

greater = x

else:

greater = y

while(True):

if((greater % x == 0) and (greater % y == 0)):

lcm = greater

break

greater += 1

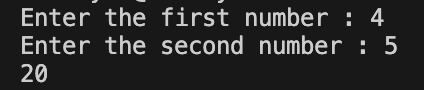
return lcm

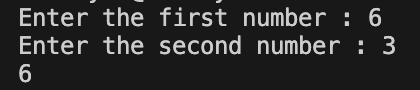
num1 = int(input("Enter the first number : "))

num2 = int(input("Enter the second number : "))

print(lcm(num1,num2))

Output:





7.Modular Exponentiation

Code:

def exponentMod(b,n,m):

if n==0:

return 1

elif n%2 == 0:

return (exponentMod(b,n/2,m)\*\*2)%m

else:

return ((exponentMod(b,n//2,m)\*\*2)%m \* b%m)%m

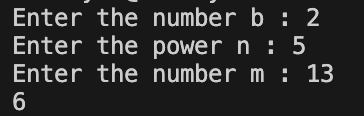
base = int(input("Enter the number b : "))

power = int(input("Enter the power n : "))

modulas = int(input("Enter the number m : "))

print(exponentMod(base,power,modulas))

Output:



8.Power of a number

Code:

def Power(a,n):

if n==0:

return 1

else:

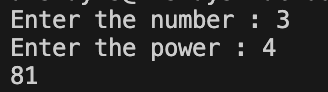
return a \* Power(a,n-1)

num = int(input("Enter the number : "))

pow = int(input("Enter the power : "))

print(Power(num,pow))

Output:



9.Fibonacci Series

Code:

def Fibonacci(n):

# Check if n is 0

# then it will return 0

if n == 0:

return 0

# Check if n is 1,2

# it will return 1

elif n == 1 or n == 2:

return 1

else:

return Fibonacci(n-1) + Fibonacci(n-2)

print("Enter a postive number to calculate the Fibonacci series ")

num=-1

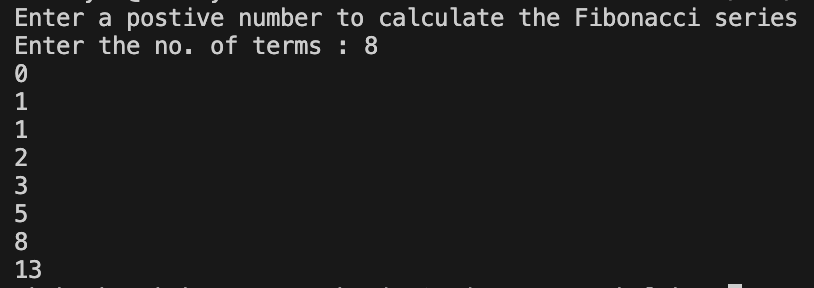
while(num<0):

num = int(input("Enter the no. of terms : "))

for i in range(num):

print(Fibonacci(i))

Output:



10.GCD Divides LCM

Code:

def gcd(a,b):

if b>a:

if a==0:

return b

else:

return gcd(b%a,a)

else:

if b==0:

return a

else:

return gcd(a%b,b)

def lcm(a,b):

return (a//gcd(a,b))\*b

def calculate\_lcm(x, y):

# selecting the greater number

if x > y:

greater = x

else:

greater = y

while(True):

if((greater % x == 0) and (greater % y == 0)):

lcm = greater

break

greater += 1

return lcm

num1 = int(input("Enter the number 1 : "))

num2 = int(input("Enter the number 2 : "))

print(lcm(num1,num2))

print(gcd(num1,num2))

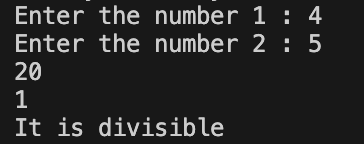
if lcm(num1,num2)%gcd(num1,num2)==0:

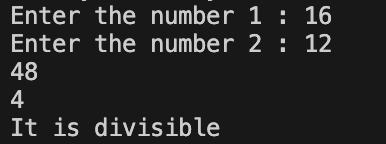
print("It is divisible")

else:

print("It is not divisible")

Output:





11.Floyd’s Algorithm:

Code:

def floyd\_warshall(graph):

n = len(graph)

dist = [row[:] for row in graph] #Or [row for row in graph]

for k in range(n):

for i in range(n):

for j in range(n):

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

return dist

# Example usage:

graph = [

[0, float('inf'), -2, float('inf')],

[4, 0, 3, float('inf')],

[float('inf'), float('inf'), 0, 2],

[float('inf'), -1, float('inf'), 0]

]

print("Original Graph (Adjacency Matrix):")

for row in graph:

print(row)

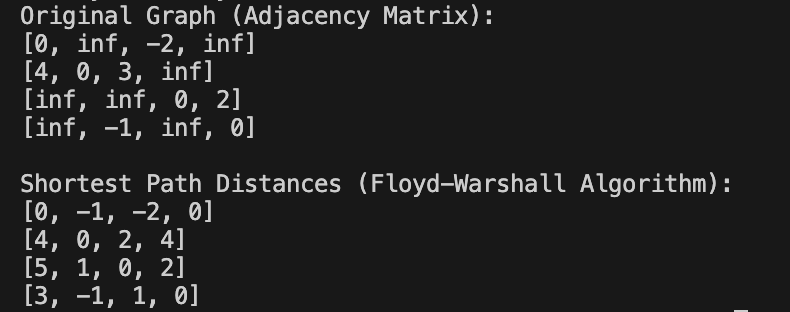
print("\nShortest Path Distances (Floyd-Warshall Algorithm):")

shortest\_paths = floyd\_warshall(graph)

for row in shortest\_paths:

print(row)

Output:



12.Warshall’s Algorithm:

Code:

def transitive\_closure(graph):

n = len(graph)

closure = [row for row in graph]

for k in range(n):

for i in range(n):

for j in range(n):

closure[i][j] = closure[i][j] or (closure[i][k] and closure[k][j])

return closure

# Example usage:

graph = [

[0, 1, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1],

[1, 0, 0, 0]

]

print("Original Graph:")

for row in graph:

print(row)

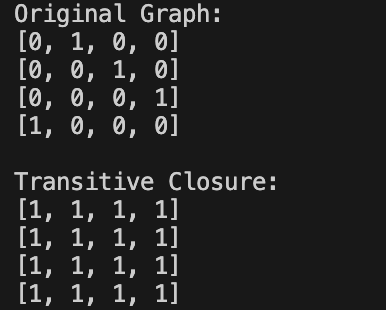
print("\nTransitive Closure:")

closure = transitive\_closure(graph)

for row in closure:

print(row)

Output:



13.Tower Of Hanoi

Code:

def tower\_of\_hanoi(n, source, auxiliary, target):

if n == 1:

print(f"Move disk 1 from {source} to {target}")

return

tower\_of\_hanoi(n-1, source, target, auxiliary)

print(f"Move disk {n} from {source} to {target}")

tower\_of\_hanoi(n-1, auxiliary, source, target)

# Example usage:

n = 3 # Number of disks

tower\_of\_hanoi(n, 'A', 'B', 'C')

Output:

