```python
## program for finding square of given number.

n1=int(input("Enter the Number : "))
if (n1>0):
    sqr = n1*n1
print(sqr)
```

```python
## Program for cube of a number.

n2=int(input("Enter the number : "))
if (n2>0):
    cube = n2*n2*n2
print(f" Cube of {n2} is {cube}")
```

```python
## Program for BFS
graph={
       'A':['C','D'],
       'B':['D','E'],
       'C':['F'],
       'D':['B'],
       'E':['G'],
       'F':[],
       'G':[]
}
visited=[]
queue=[]
def bfs(visited,graph,node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print (s, end = " ")

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, 'A')
```

```python
## Program for BFS
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
    visited.append(node)
    queue.append(node)

    while queue:           # Creating loop to visit each node
        m = queue.pop(0)
        print (m, end = " ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

```python
## program for factorial of a number
def fact(n):
    if (n==0) or (n==1):
        return 1
    else:
        return n*fact(n-1)
num=int(input("Enter the Number : "))
print(f"The Factorial of {num} is : {fact(num)}")
```

```python
## program to find area of a triangle
def area(b,h):
    if (b==0) or (h==0):
        return 0
    else:
        return (1/2*b*h)
area(4,6)
```

```python
## program to find area of a circle
```

```python
from math import pi
def cir_area(r):
    if r==0:
        return 0
    else:
        return pi*r*r
cir_area(4)
```

```python
## Program to find area of rectangle
def rect(l,b):
    if (l==0) or (b==0):
        return 0
    else:
        return (l*b)
rect(2,3)
```

```python
## Program for DFS
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

```python
# LCM Program
def lcm(a,b):
    big=max(a,b)
    small=min(a,b)
    for i in range(big,a*b+1,big):
        if i%small==0:
            return i
```

```
a=int(input("Enter First NUmber : "))
b=int(input("Enter Second NUmber : "))
print(f"LCM of {a} and {b} is : {lcm(a,b)}")
```

```
# Palindrome
def palin(n):
    temp=n
    rev=0
    while(temp>0):
        rem=temp % 10
        rev=rev*10+rem
        temp=temp // 10
    if(n == rev):
        print(f"{n} is a Palindrome Number")
    else:
        print(f"{n} is not palindrome Number")
n=int(input("Enter the Number : "))
print(palin(n))
```

```
print(1234//10)
```
output:123

```
def pattern(n):
    for i in range(1,n):
        for i in range(1,i+1):
            print("A",end=" ")
        print()
pattern(5)
```
```
A
A A
A A A
A A A A
```

```
def perf_num(n):
    sum=0
    for i in range(1,n):
        if(n%i == 0):
            sum=sum+i
    if(sum == n):
        print(f"{n} is a Perfect Number.")
    else:
        print(f"{n} is not a Perfect Number.")
n=int(input("Enter the Number : "))
print(perf_num(n))
```

```
Enter the Number : 6
6 is a Perfect Number.
None
```

```python
graph={
    '5':['6','7'],
    '6':['4','7'],
    '7':['3','4'],
    '4':[],
    '3':[]
}
visited=set()
def dfs(visited,graph,node):
    if node not in visited:
        print(node)
        visited.add(node)
    for neighbour in graph[node]:
        dfs(visited,graph,neighbour)
print("The DFS Traversal is :")
dfs(visited,graph,'5')
```

```
The DFS Traversal is :
5
6
4
7
3
```

```python
graph={
    '5':['6','7'],
    '6':['3','4'],
    '7':['2'],
    '3':[],
    '4':[],
    '2':['8'],
    '8':[]
}
visited=[]
queue=[]
def bfs(visited,graph,node):
    visited.append(node)
    queue.append(node)
    while queue:
        k=queue.pop(0)
        print(k,end=" ")
        for neighbour in graph[k]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("The BFS Traversal is : ")
bfs(visited,graph,'5')
```

```
The BFS Traversal is :
5 6 7 3 4 2 8
```

```python
from collections import deque
def BFS(a, b, target1, target2):
    # Track visited states and the path to the solution
    visited =set()
    path = []
    queue = deque([(0, 0)])  # Start with both jugs empty

    while queue:
        current = queue.popleft()
        if current in visited:
            continue
        visited.add(current)
        path.append(current)

        x, y = current
        d = 0
        # Check if we have reached the target
        if x == target1 and y == target2:
            '''if x == target and y != 0:
                path.append((x, 0))
            if y == target and x != 0:
                path.append((0, y))'''
            for p in path:
                print(f"({p[0]}, {p[1]})")
            return

        # Generate all possible next states
        next_states = [
            #(a, y),  # Fill Jug1
            (x, b),  # Fill Jug2
            (x - min(x, b - y), y + min(x, b - y)),  # Pour Jug1 to Jug2
            (x + min(y, a - x),
             y - min(y, a - x)),  # Pour Jug2 to Jug1
            (x - d, y),
            (x, y - d),
            (0, y),  # Empty Jug1
            #(x, 0)  # Empty Jug2
        ]

        for state in next_states:
            if state not in visited:
                #visited.add(state)
                queue.append(state)

    print("No solution")
Jug1, Jug2, t1, t2 = 7, 9, 2, 0
print("Path from initial state to solution state")
BFS(Jug1, Jug2, t1, t2)
```

```
Path from initial state to solution state
(0, 0)
(0, 9)
(7, 2)
(7, 9)
(0, 2)
(2, 0)
```

```python
#Program for finding GCD of two numbers
def gcd(a, b):
    while b != 0:  # Continue until the remainder is 0
        a,b = b, a % b  # Update a with b, and b with the remainder of a
divided by b
    return a  # When b becomes 0, a contains the GCD

# Example usage
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

result = gcd(num1, num2)
print(f"The GCD of {num1} and {num2} is {result}.")
```

```
Enter the first number: 4
Enter the second number: 5
The GCD of 4 and 5 is 1.
```

```python
print(4%6)
```

```
4
```