Programming for Problem Solving

Reinprep Problem Solutions

Week 1 (10 Questions)

Week 1.1 Two Sum

Problem Statement:

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order

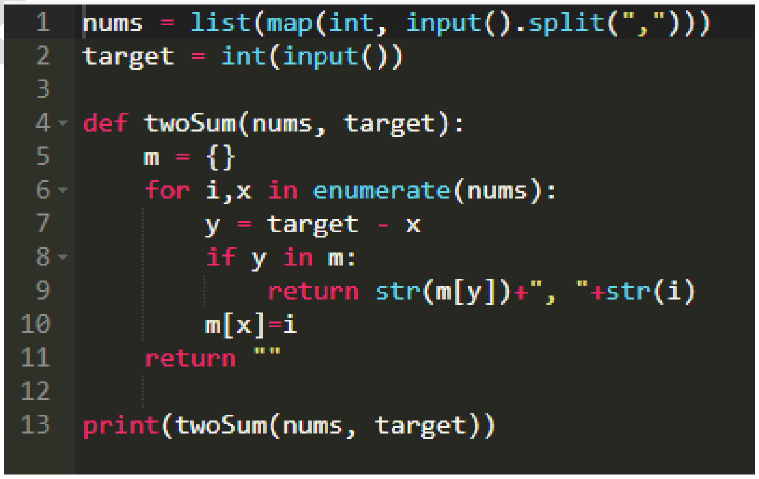
Input:

2, 7, 11, 15
9

Output:

0, 1

Explanation: Because nums[0] + nums[1] == 9, so return [0, 1].



```python
nums = list(map(int, input().split(",")))
target = int(input())

def twoSum(nums, target):
    m = {}
    for i,x in enumerate(nums):
        y = target - x
        if y in m:
            return str(m[y])+", "+str(i)
        m[x]=i
    return ""

print(twoSum(nums, target))
```

Week 1.2 Contains Duplicate

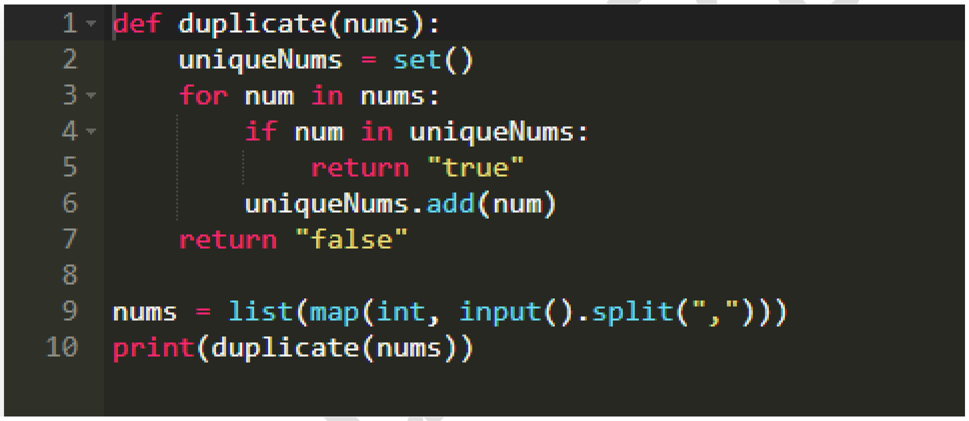
Problem Statement:
Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Input:

1, 2, 3, 1

Output:

True



```python
def duplicate(nums):
    uniqueNums = set()
    for num in nums:
        if num in uniqueNums:
            return "true"
        uniqueNums.add(num)
    return "false"

nums = list(map(int, input().split(",")))
print(duplicate(nums))
```

Week 1.3 Roman to Integer

Problem Statement:

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol Value

I 1, V 5, X 10, L 50, C 100, D 500, M 1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

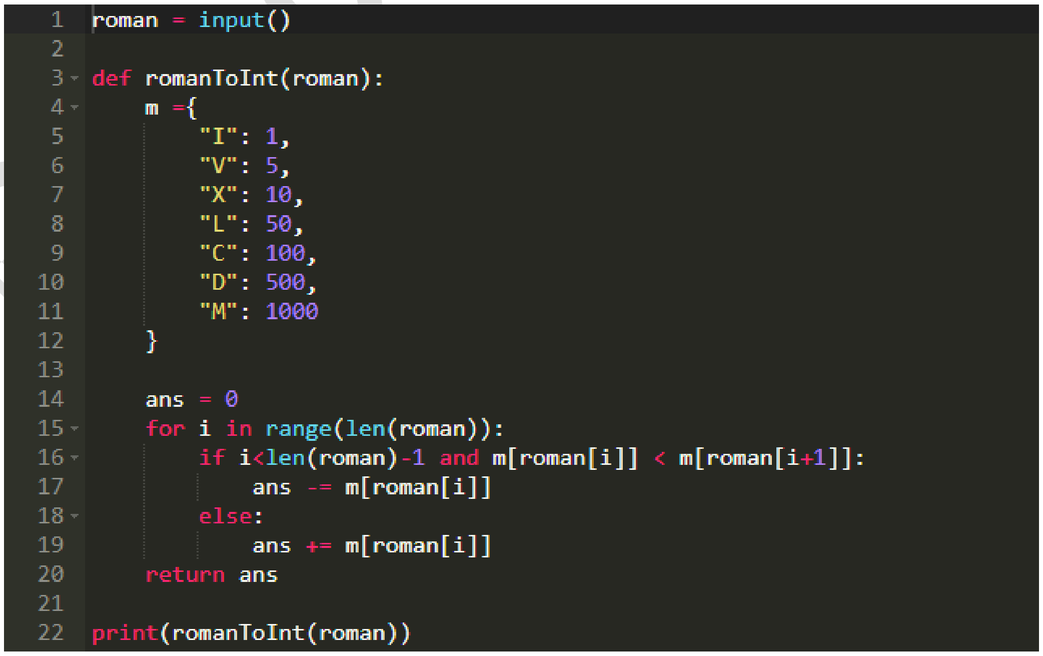
Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

Input:

III

Output:

3



```python
roman = input()

def romanToInt(roman):
    m ={
        "I": 1,
        "V": 5,
        "X": 10,
        "L": 50,
        "C": 100,
        "D": 500,
        "M": 1000
    }

    ans = 0
    for i in range(len(roman)):
        if i<len(roman)-1 and m[roman[i]] < m[roman[i+1]]:
            ans -= m[roman[i]]
        else:
            ans += m[roman[i]]
    return ans

print(romanToInt(roman))
```

Week 1.4 Plus One

Problem Statement:

You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's. Increment the large integer by one and return the resulting array of digits.

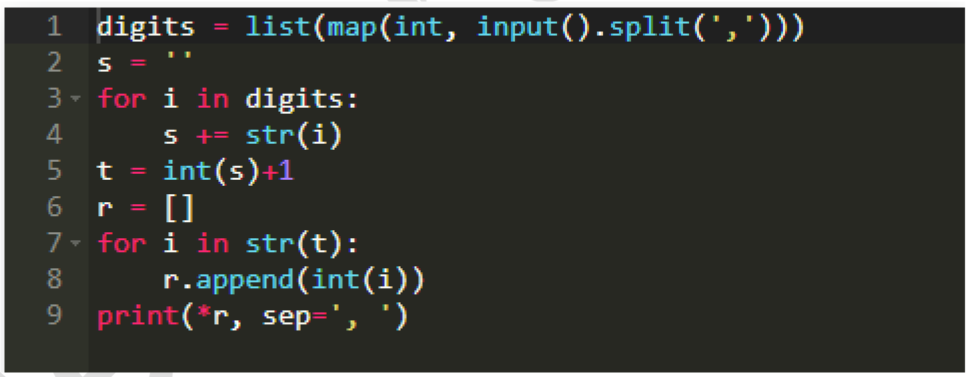
Input:

1, 2, 3

Output:

1, 2, 4

Explanation: The array represents the integer 123. Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1, 2, 4].



```python
digits = list(map(int, input().split(',')))
s = ''
for i in digits:
    s += str(i)
t = int(s)+1
r = []
for i in str(t):
    r.append(int(i))
print(*r, sep=', ')
```

Week 1.5 Majority Element

Problem Statement:

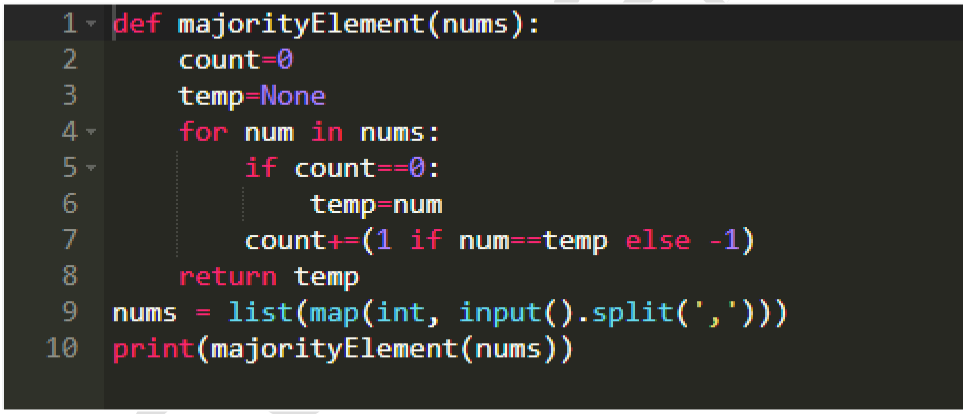
Given an array nums of size n, return the majority element. The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Input:

3, 2, 3

Output:

3



```python
def majorityElement(nums):
    count=0
    temp=None
    for num in nums:
        if count==0:
            temp=num
        count+=(1 if num==temp else -1)
    return temp
nums = list(map(int, input().split(',')))
print(majorityElement(nums))
```

Week 1.6 Richest Customer Wealth

Problem Statement:
You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return the wealth that the richest customer has. A customer's wealth is the amount of money they have in all their bank accounts.

Input format: First line reads number of customers, subsequent lines indicates the amount in the different banks of the customer

Output format: The total wealth of the richest customer

Input:

2

1, 2, 3

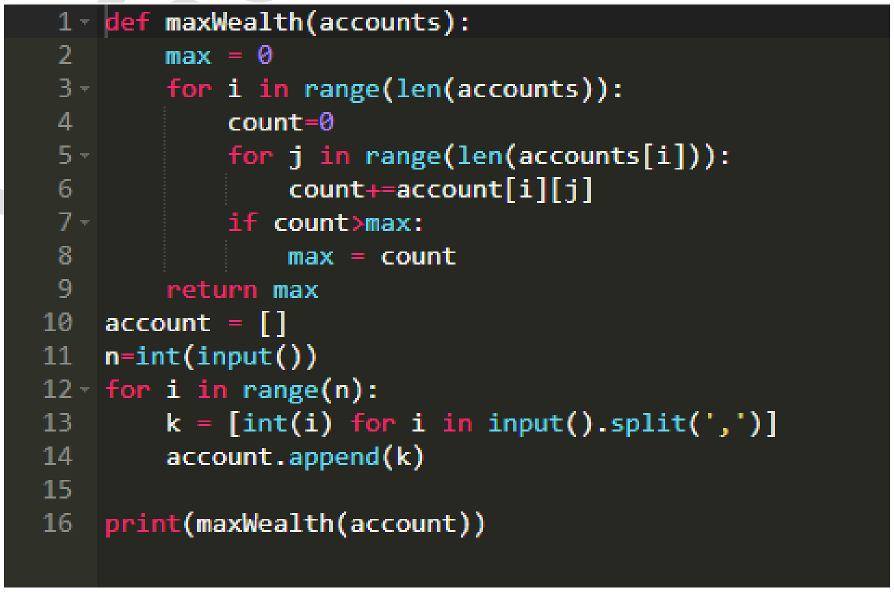
3, 2, 5

Output:

10

Explanation: 1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 5 = 10



```
def maxWealth(accounts):
    max = 0
    for i in range(len(accounts)):
        count=0
        for j in range(len(accounts[i])):
            count+=account[i][j]
        if count>max:
            max = count
    return max
account = []
n=int(input())
for i in range(n):
    k = [int(i) for i in input().split(',')]
    account.append(k)

print(maxWealth(account))
```

Week 1.7 Fizz Buzz

Problem Statement:
Given an integer n, return a string array answer (1-indexed) where:

answer[i] == "FizzBuzz" if i is divisible by 3 and 5.

answer[i] == "Fizz" if i is divisible by 3.

answer[i] == "Buzz" if i is divisible by 5.

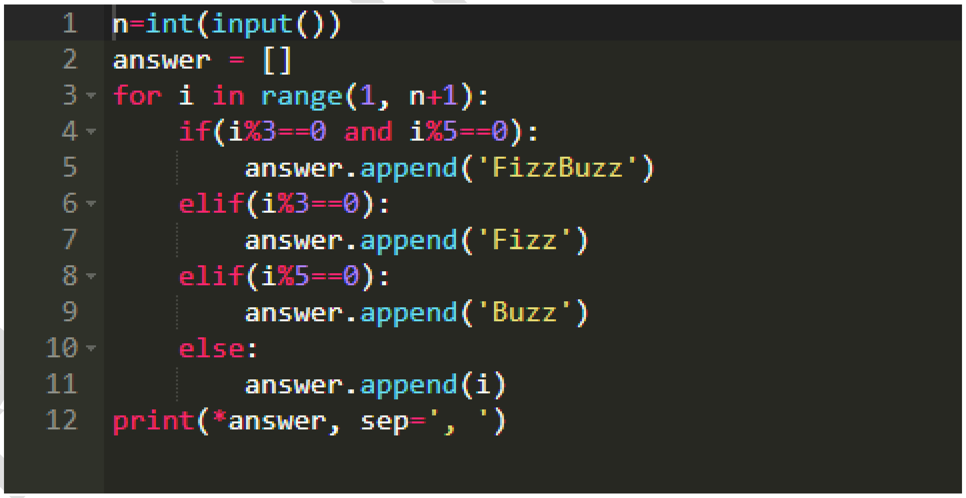
answer[i] == i (as a string) if none of the above conditions are true.

Input:

3

Output:

1, 2, Fizz



```python
n=int(input())
answer = []
for i in range(1, n+1):
    if(i%3==0 and i%5==0):
        answer.append('FizzBuzz')
    elif(i%3==0):
        answer.append('Fizz')
    elif(i%5==0):
        answer.append('Buzz')
    else:
        answer.append(i)
print(*answer, sep=', ')
```

Week 1.8 Number of steps to reduce a number to zero

Problem Statement:
Given an integer num, return the number of steps to reduce it to zero. In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

Input:

14

Output:

6

Explanation: •

14 is even; divide by 2 and obtain 7. •

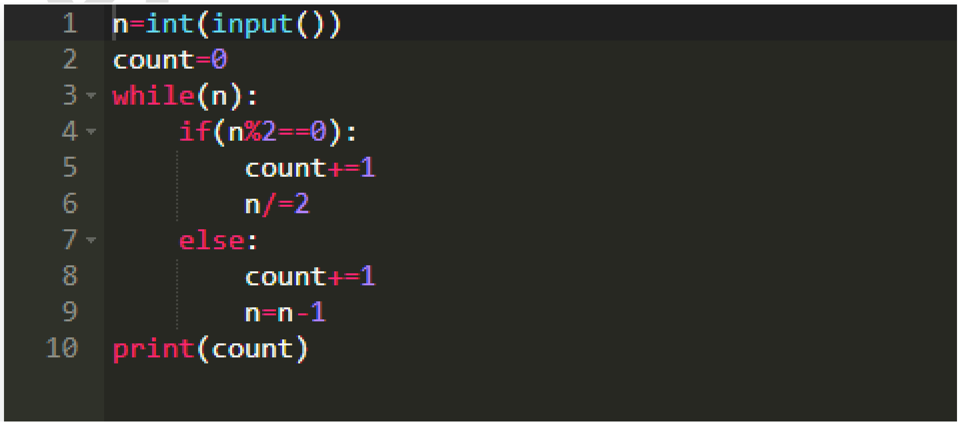
7 is odd; subtract 1 and obtain 6. •

6 is even; divide by 2 and obtain 3. •

3 is odd; subtract 1 and obtain 2. •

2 is even; divide by 2 and obtain 1. •

1 is odd; subtract 1 and obtain 0.



```python
n=int(input())
count=0
while(n):
    if(n%2==0):
        count+=1
        n/=2
    else:
        count+=1
        n=n-1
print(count)
```

Week 1.9 Running sum of 1D array

Problem Statement:
Given an array nums. We define a running sum of an array as runningSum[i] = sum (nums[0]…nums[i]). Return the running sum of nums.

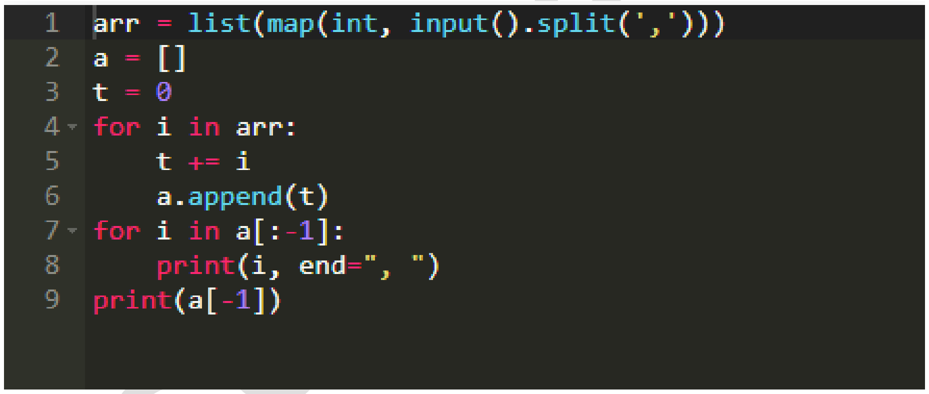
Input:

1, 2, 3, 4

Output:

1, 3, 6, 10

Explanation: Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].



```python
arr = list(map(int, input().split(',')))
a = []
t = 0
for i in arr:
    t += i
    a.append(t)
for i in a[:-1]:
    print(i, end=", ")
print(a[-1])
```

Week 1.10 Remove Element

Problem Statement:
Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val. Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.
- Return k.

Input:

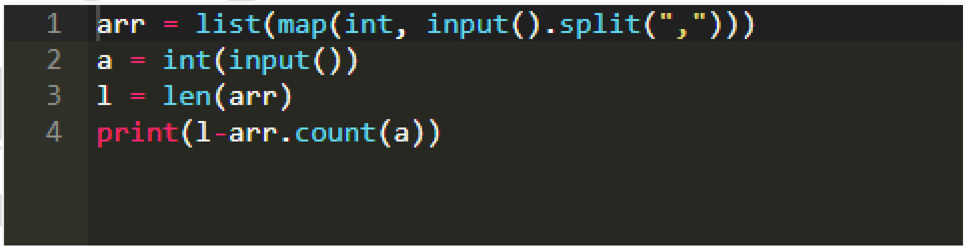
3, 2, 2, 3

3

Output:

2

Explanation: Your function should return k = 2, with the first two elements of nums being 2. It does not matter what you leave beyond the returned k (hence they are underscores).



```python
arr = list(map(int, input().split(",")))
a = int(input())
l = len(arr)
print(l-arr.count(a))
```

Week 2 (4 Questions)

Week 2.1 Add Two Matrices

Problem Statement:
Given two matrices X and Y, the task is to compute the sum of two matrices and then print it in Python.

Input:

2

1, 2

4, 5

2

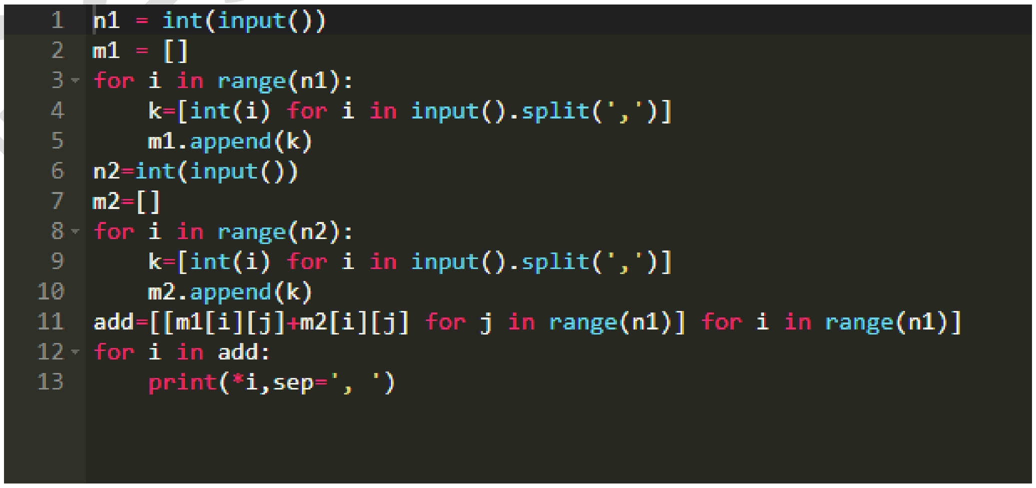
9, 8

6, 5

Output:

10, 10

10, 10



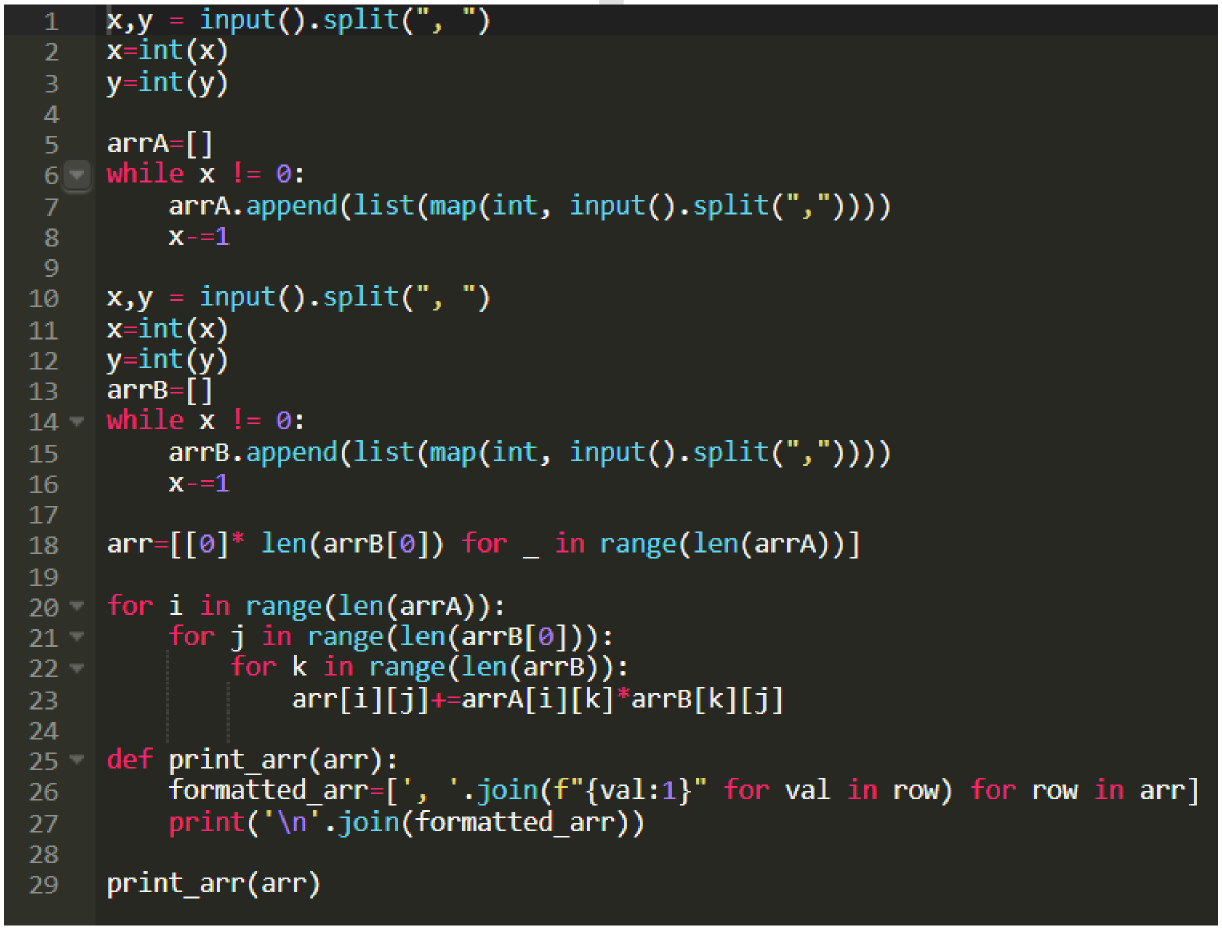
```python
n1 = int(input())
m1 = []
for i in range(n1):
    k=[int(i) for i in input().split(',')]
    m1.append(k)
n2=int(input())
m2=[]
for i in range(n2):
    k=[int(i) for i in input().split(',')]
    m2.append(k)
add=[[m1[i][j]+m2[i][j] for j in range(n1)] for i in range(n1)]
for i in add:
    print(*i,sep=', ')
```

Week 2.2 Multiply Two Matrices

Problem Statement:
Given two matrices X and Y, the task is to compute the multiplication of two matrices and then print it.

Input:

matrix X	**matrix Y**	**Output:**
3, 3	3, 4	55, 65, 49, 5
1, 7, 3	1, 1, 1, 2	57, 68, 72, 12
3, 5, 6	6, 7, 3, 0	90, 107, 111, 21
6, 8, 9	4, 5, 9, 1	



```python
x,y = input().split(", ")
x=int(x)
y=int(y)

arrA=[]
while x != 0:
    arrA.append(list(map(int, input().split(","))))
    x-=1

x,y = input().split(", ")
x=int(x)
y=int(y)
arrB=[]
while x != 0:
    arrB.append(list(map(int, input().split(","))))
    x-=1

arr=[[0]* len(arrB[0]) for _ in range(len(arrA))]

for i in range(len(arrA)):
    for j in range(len(arrB[0])):
        for k in range(len(arrB)):
            arr[i][j]+=arrA[i][k]*arrB[k][j]

def print_arr(arr):
    formatted_arr=[', '.join(f"{val:1}" for val in row) for row in arr]
    print('\n'.join(formatted_arr))

print_arr(arr)
```

Week 2.3 Transpose of a Matrix

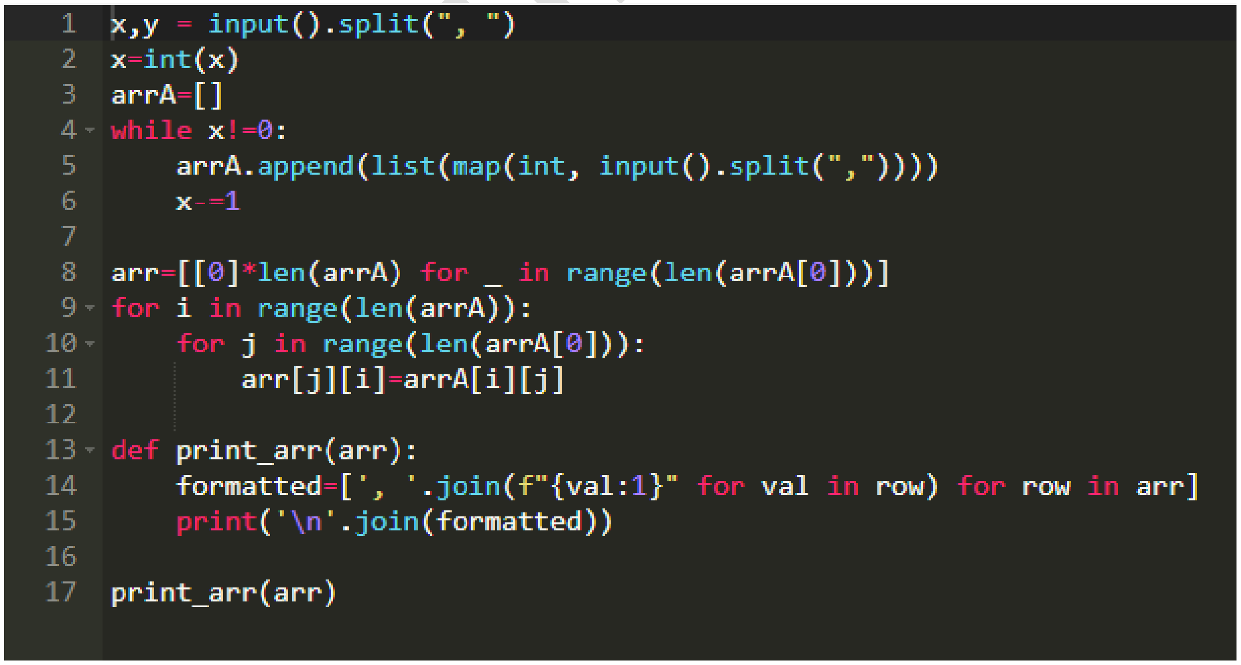
Problem Statement:
A matrix can be implemented using a nested list. Each element is treated as a row of the matrix. Find the transpose of a matrix in multiple ways.

Input:	**Output:**
3, 2	1, 3, 5
1, 2	2, 4, 6
3, 4	
5, 6	

Explanation:

Suppose we are given a matrix [[1, 2], [3, 4], [5, 6]]

Then the transpose of the given matrix will be, [[1, 3, 5], [2, 4, 6]]



```python
x,y = input().split(", ")
x=int(x)
arrA=[]
while x!=0:
    arrA.append(list(map(int, input().split(","))))
    x-=1

arr=[[0]*len(arrA) for _ in range(len(arrA[0]))]
for i in range(len(arrA)):
    for j in range(len(arrA[0])):
        arr[j][i]=arrA[i][j]

def print_arr(arr):
    formatted=[', '.join(f"{val:1}" for val in row) for row in arr]
    print('\n'.join(formatted))

print_arr(arr)
```

Week 2.4 Matrix Product

Problem Statement:
Matrix product problem we can solve using list comprehension as a potential shorthand to the conventional loops. Iterate and find the product of the nested list and at the end return the cumulative product using function.

Input:

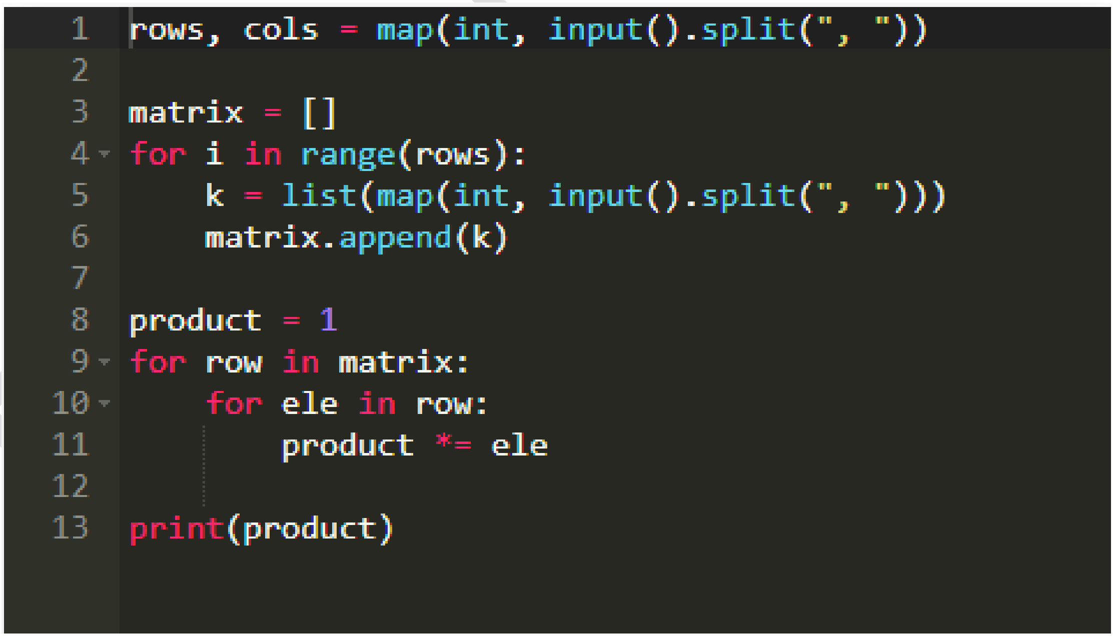
3, 3
1, 4, 5

7, 3, 4

46, 7, 3

Output:

1622880



```python
rows, cols = map(int, input().split(", "))

matrix = []
for i in range(rows):
    k = list(map(int, input().split(", ")))
    matrix.append(k)

product = 1
for row in matrix:
    for ele in row:
        product *= ele

print(product)
```

Week 3 (2 Questions)

Week 3.1 Balanced Paranthesis Checking

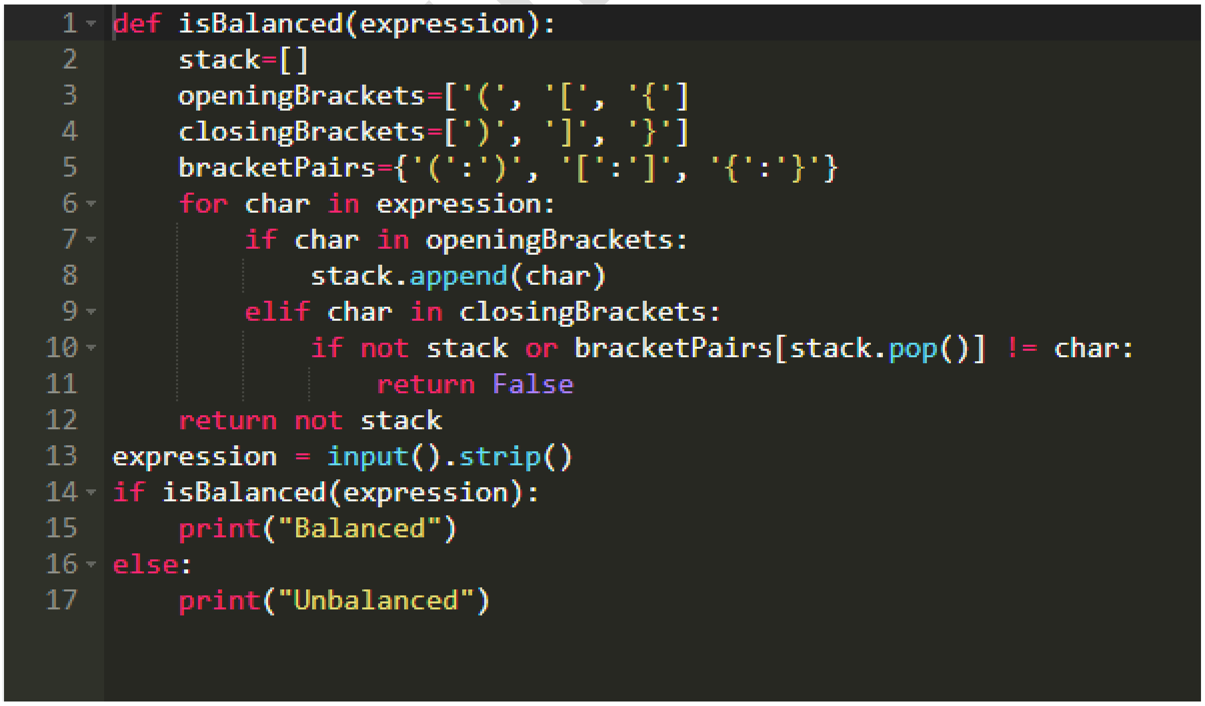
Problem Statement:
Given an expression string, write a python program to find whether a given string has balanced parentheses or not.

Input: {[]{()}}

Output: Balanced

Explanation:

Using stack One approach to check balanced parentheses is to use stack. Each time, when an open parentheses is encountered push it in the stack, and when closed parenthesis is encountered, match it with the top of stack and pop it. If stack is empty at the end, return Balanced otherwise, Unbalanced.



```python
def isBalanced(expression):
    stack=[]
    openingBrackets=['(', '[', '{']
    closingBrackets=[')', ']', '}']
    bracketPairs={'(':')', '[':']', '{':'}'}
    for char in expression:
        if char in openingBrackets:
            stack.append(char)
        elif char in closingBrackets:
            if not stack or bracketPairs[stack.pop()] != char:
                return False
    return not stack
expression = input().strip()
if isBalanced(expression):
    print("Balanced")
else:
    print("Unbalanced")
```

Week 3.2 Evaluation of Postfix Expression

Problem Statement:
Given a postfix expression, the task is to evaluate the postfix expression. Postfix expression: The expression of the form “a b operator” (ab+) i.e., when a pair of operands is followed by an operator.

The allowed arithmetic operators are + - * /(float division rounded upto single decimal digit)

Input format: the operand and operator must be separated by single space

Output format: The result is Numerical

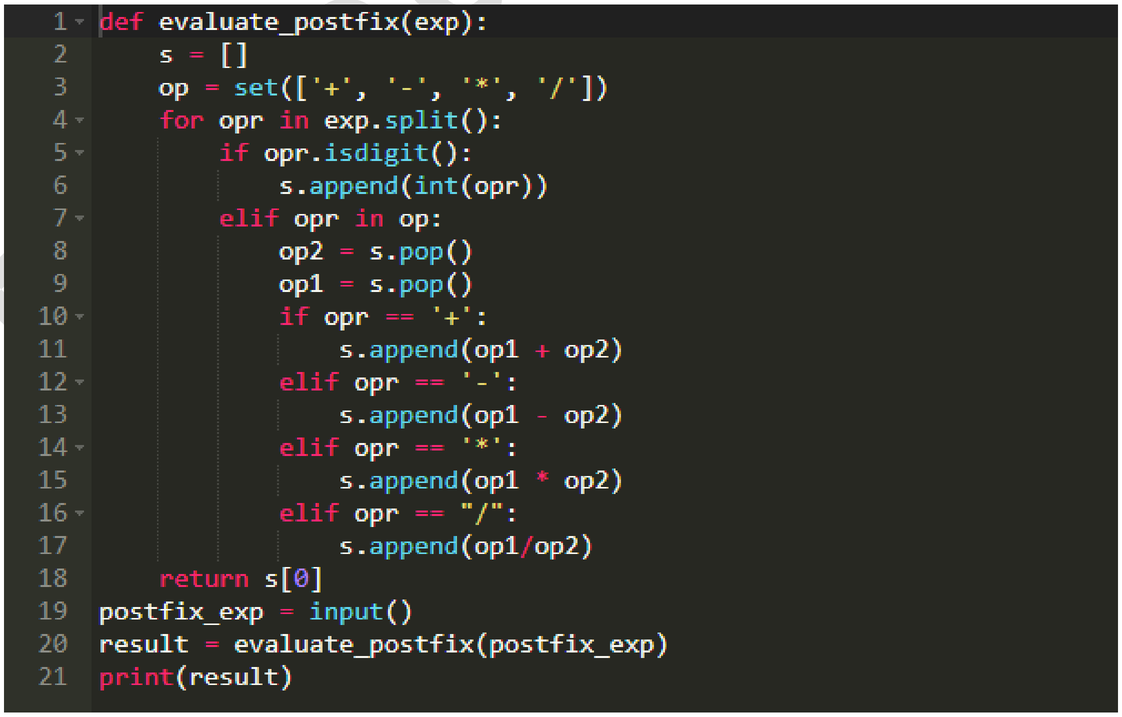
Input:

2 3 1 * + 9 -

Output:

-4

Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.



```python
def evaluate_postfix(exp):
    s = []
    op = set(['+', '-', '*', '/'])
    for opr in exp.split():
        if opr.isdigit():
            s.append(int(opr))
        elif opr in op:
            op2 = s.pop()
            op1 = s.pop()
            if opr == '+':
                s.append(op1 + op2)
            elif opr == '-':
                s.append(op1 - op2)
            elif opr == '*':
                s.append(op1 * op2)
            elif opr == "/":
                s.append(op1/op2)
    return s[0]
postfix_exp = input()
result = evaluate_postfix(postfix_exp)
print(result)
```

Week 4 (4 Questions)

Week 4.1 Linear Queue using List

Problem Statement:
Linear queue is a linear data structure that stores items in First in First out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first

Input:

6

add, add, add, size, dequeue, print

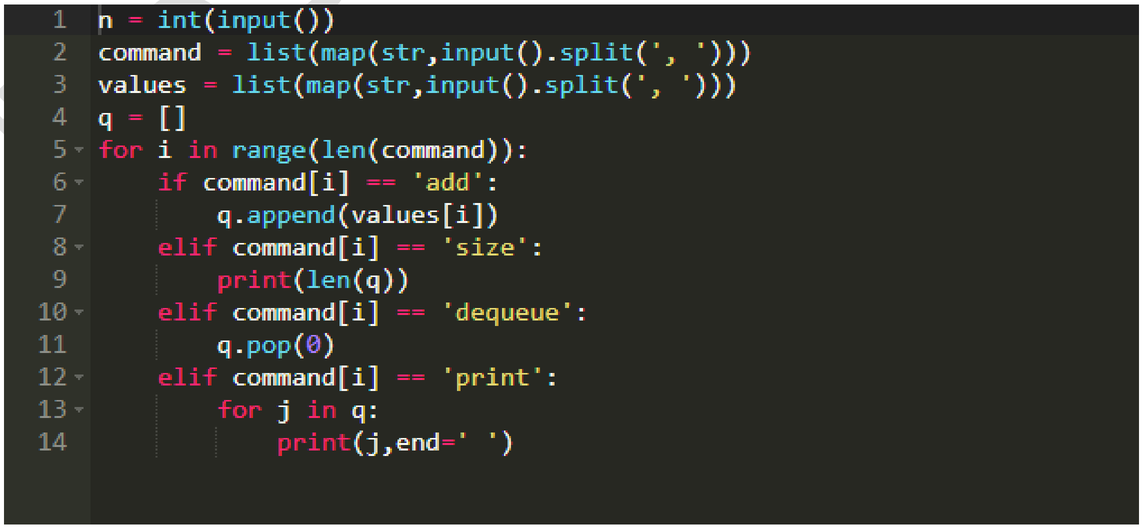
1, 3, 4, NULL, NULL, NULL

Sample Output:

3

3 4

Explanation: 1, 3, and 4 are added. Then size is printed (3). Dequeue removes (1) and then prints the array.



```python
n = int(input())
command = list(map(str,input().split(', ')))
values = list(map(str,input().split(', ')))
q = []
for i in range(len(command)):
    if command[i] == 'add':
        q.append(values[i])
    elif command[i] == 'size':
        print(len(q))
    elif command[i] == 'dequeue':
        q.pop(0)
    elif command[i] == 'print':
        for j in q:
            print(j,end=' ')
```

Week 4.2 Stack using Queues

Problem Statement:
Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

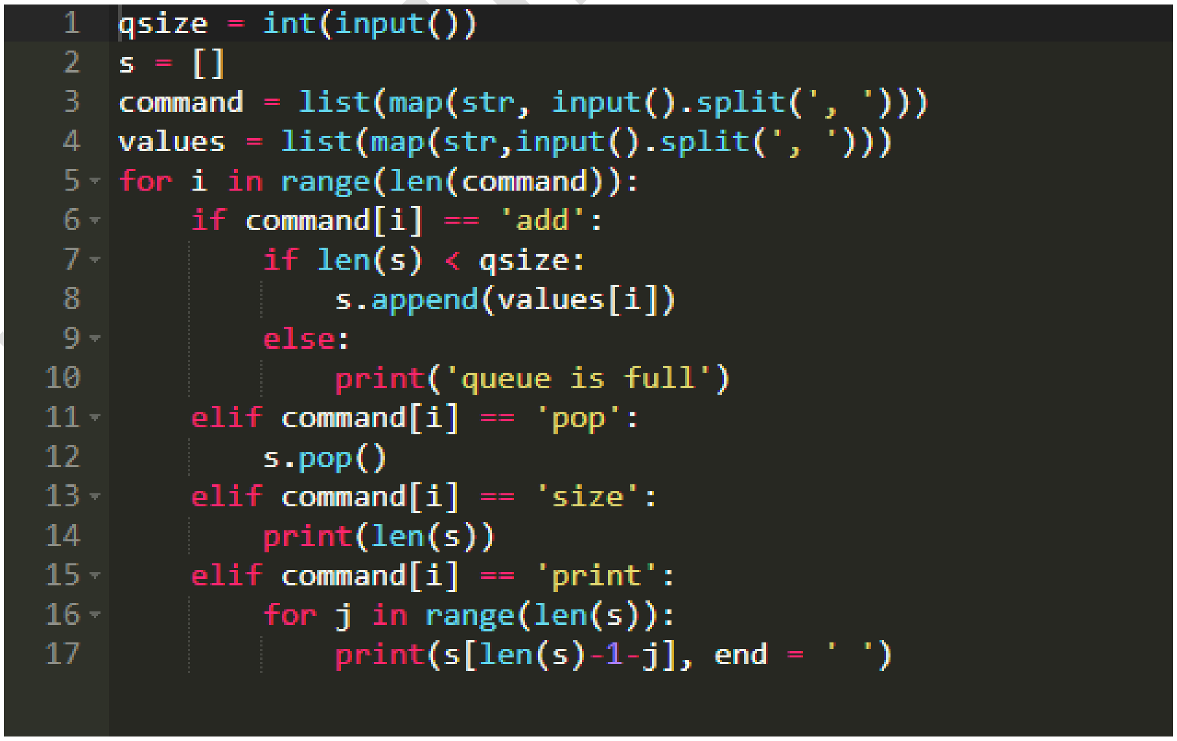
• void push(int x) Pushes element x to the top of the stack.

• int pop() Removes the element on the top of the stack and returns it.

• int top() Returns the element on the top of the stack.

• boolean empty() Returns true if the stack is empty, false otherwise.

Input:	**Output:**
6	1
add, pop, add, size, print, pop	3
1, NULL, 3, NULL, NULL, NULL	



```python
qsize = int(input())
s = []
command = list(map(str, input().split(', ')))
values = list(map(str,input().split(', ')))
for i in range(len(command)):
    if command[i] == 'add':
        if len(s) < qsize:
            s.append(values[i])
        else:
            print('queue is full')
    elif command[i] == 'pop':
        s.pop()
    elif command[i] == 'size':
        print(len(s))
    elif command[i] == 'print':
        for j in range(len(s)):
            print(s[len(s)-1-j], end = ' ')
```

Week 4.3 Generate Binary Numbers from 1 to n

Problem Statement:
Given a number N, write a function that generates and prints all binary numbers with decimal values from 1 to N.

if n is 0 return 0

else print the binary number equivalent from 1 to n

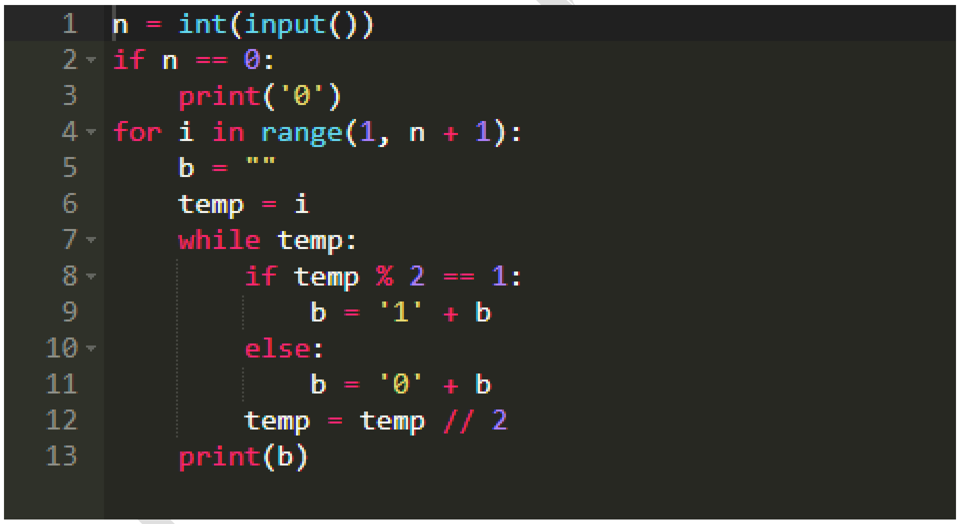
Examples:

Input: n = 2

Output:

1

10



```python
n = int(input())
if n == 0:
    print('0')
for i in range(1, n + 1):
    b = ""
    temp = i
    while temp:
        if temp % 2 == 1:
            b = '1' + b
        else:
            b = '0' + b
        temp = temp // 2
    print(b)
```

Week 4.4 Next Greater Element for every element in given Array

Problem Statement:
Given an array, print the Next Greater Element (NGE) for every element.

The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Example:

Input: 4 , 5 , 2 , 25

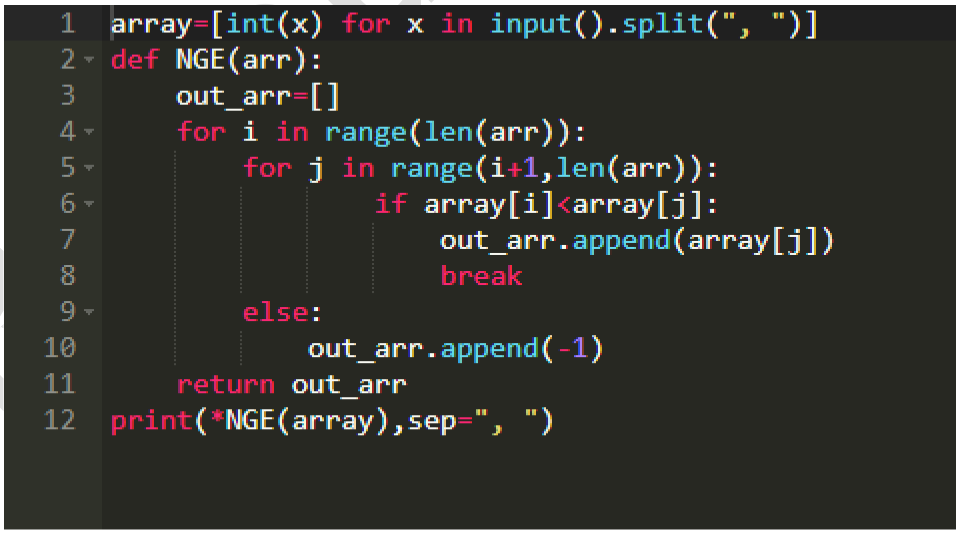
Output: 4 –> 5

5 -> 25

2 –> 25

25 –> -1

Output format: 5, 25, 25, -1



```python
array=[int(x) for x in input().split(", ")]
def NGE(arr):
    out_arr=[]
    for i in range(len(arr)):
        for j in range(i+1,len(arr)):
            if array[i]<array[j]:
                out_arr.append(array[j])
                break
        else:
            out_arr.append(-1)
    return out_arr
print(*NGE(array),sep=", ")
```

Week 5 (5 Questions)

Week 5.1 Build a Graph: Representations of Graph

Problem:
Given the number of vertices, number of edges and edges of the graph, the task is to represent the adjacency list for an undirected graph.

Input format: first line contains number of vertices, number of edges (comma separated) subsequent lines will give the edges of the graph

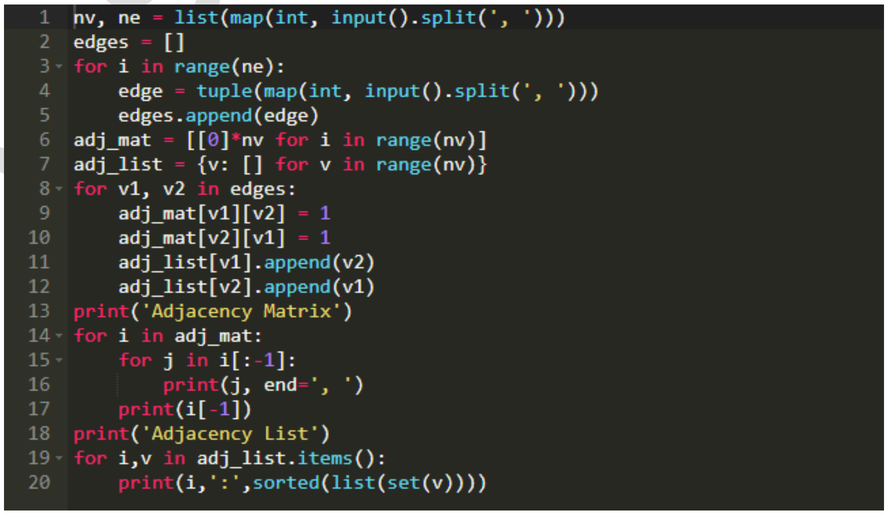
Output format:
First line displays the string "Adjacency Matrix"
subsequent lines gives each row of the adjacency matrix
The next line displays string "Adjacency List"
the next lines gives each vertex with adjacent vertices

Input:

```
3, 3
0, 1
1, 2
2, 0
```

Output:

```
Adjacency Matrix        Adjacency List
0, 1, 1                 0 : [1, 2]
1, 0, 1                 1 : [0, 2]
1, 1, 0                 2 : [0, 1]
```



```python
nv, ne = list(map(int, input().split(', ')))
edges = []
for i in range(ne):
    edge = tuple(map(int, input().split(', ')))
    edges.append(edge)
adj_mat = [[0]*nv for i in range(nv)]
adj_list = {v: [] for v in range(nv)}
for v1, v2 in edges:
    adj_mat[v1][v2] = 1
    adj_mat[v2][v1] = 1
    adj_list[v1].append(v2)
    adj_list[v2].append(v1)
print('Adjacency Matrix')
for i in adj_mat:
    for j in i[:-1]:
        print(j, end=', ')
    print(i[-1])
print('Adjacency List')
for i,v in adj_list.items():
    print(i,':',sorted(list(set(v))))
```

Week 5.2 Number of Sink nodes in a Graph

Problem Statement:

Given a Directed Acyclic Graph of n nodes (numbered from 1 to n) and m edges. The task is to find the number of sink nodes. A sink node is a node such that no edge emerges out of it.

Input Format:

- The first line contains an integer n, the number of nodes
- The second line contains an integer m, the number of edges
- The next m lines contain two space seperated integers a and b which indicate an edge between the nodes a and b.

Example:

Input:

4

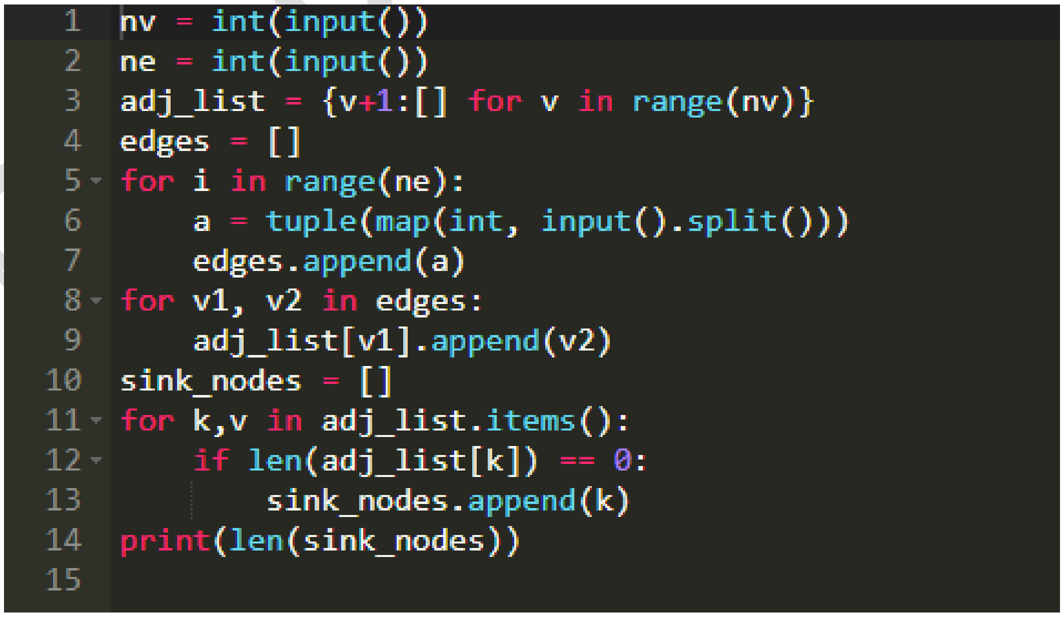
2

2 3

4 3

Output:

2



```
nv = int(input())
ne = int(input())
adj_list = {v+1:[] for v in range(nv)}
edges = []
for i in range(ne):
    a = tuple(map(int, input().split()))
    edges.append(a)
for v1, v2 in edges:
    adj_list[v1].append(v2)
sink_nodes = []
for k,v in adj_list.items():
    if len(adj_list[k]) == 0:
        sink_nodes.append(k)
print(len(sink_nodes))
```

Week 5.3 Connected Components in a Graph

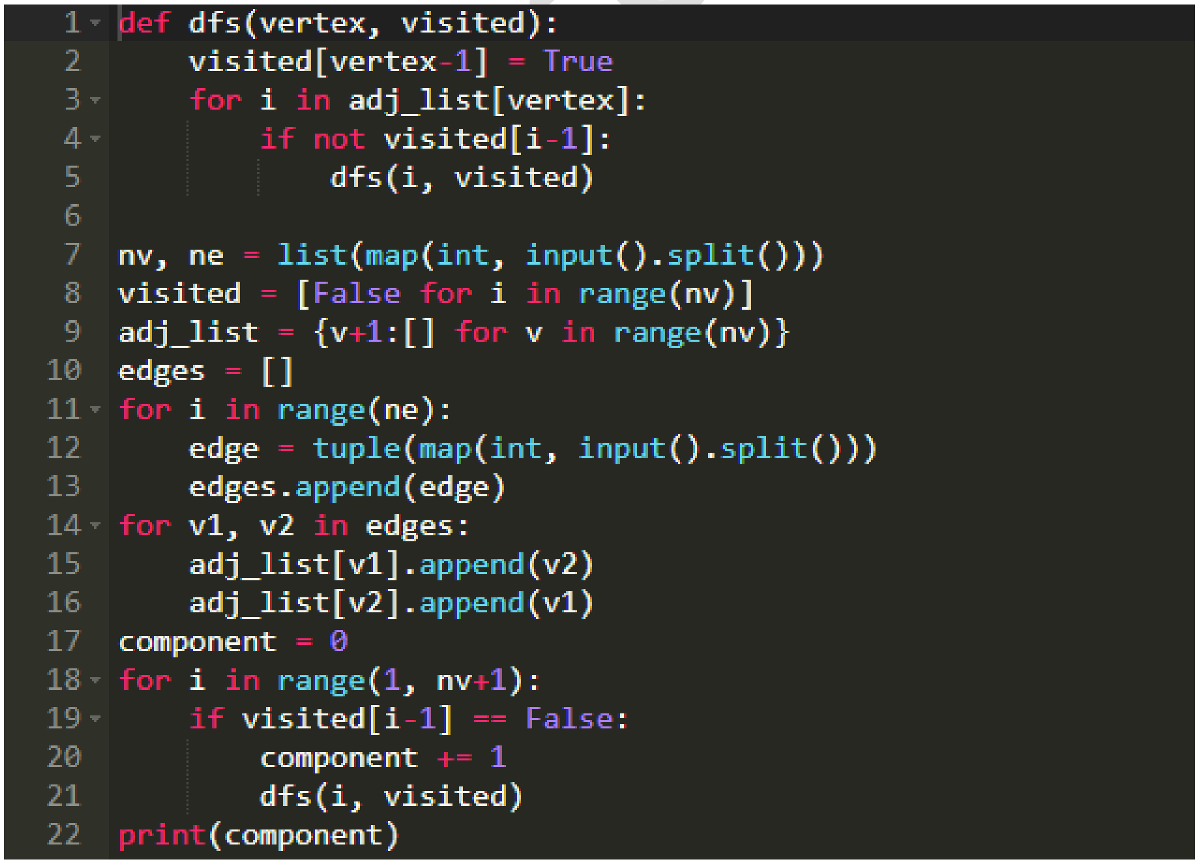
Problem Statement:
Given n, i.e. total number of nodes in an undirected graph numbered from 1 to n and an integer e, i.e. total number of edges in the graph. Calculate the total number of connected components in the graph. A connected component is a set of vertices in a graph that are linked to each other by paths.

Input:

First line of input line contains two integers' n and e. Next e line will contain two integers u and v meaning that node u and node v are connected to each other in undirected fashion.

Output:

For each input graph print an integer x denoting total number of connected components.



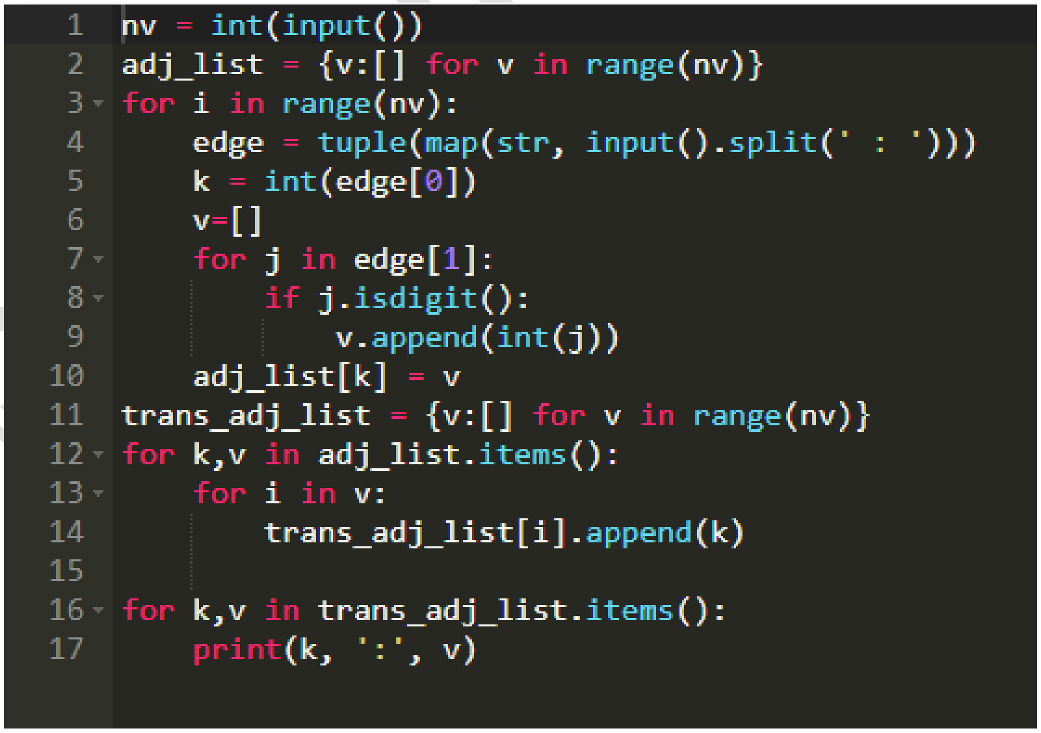
```python
def dfs(vertex, visited):
    visited[vertex-1] = True
    for i in adj_list[vertex]:
        if not visited[i-1]:
            dfs(i, visited)

nv, ne = list(map(int, input().split()))
visited = [False for i in range(nv)]
adj_list = {v+1:[] for v in range(nv)}
edges = []
for i in range(ne):
    edge = tuple(map(int, input().split()))
    edges.append(edge)
for v1, v2 in edges:
    adj_list[v1].append(v2)
    adj_list[v2].append(v1)
component = 0
for i in range(1, nv+1):
    if visited[i-1] == False:
        component += 1
        dfs(i, visited)
print(component)
```

Week 5.4 Transpose Graph

Problem Statement:
Transpose of a directed graph G is another directed graph on the same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in G. That is, if G contains an edge (u, v) then the converse/transpose/reverse of G contains an edge (v, u) and vice versa. Given a graph (represented as adjacency list), we need to find another graph which is the transpose of the given graph.

Example:

Input:	**Output:**
4	0 : []
0 : 1	1 : [0]
1 : 2	2 : [1]
2 : 3	3 : [2]
3 :	



```python
nv = int(input())
adj_list = {v:[] for v in range(nv)}
for i in range(nv):
    edge = tuple(map(str, input().split(' : ')))
    k = int(edge[0])
    v=[]
    for j in edge[1]:
        if j.isdigit():
            v.append(int(j))
    adj_list[k] = v
trans_adj_list = {v:[] for v in range(nv)}
for k,v in adj_list.items():
    for i in v:
        trans_adj_list[i].append(k)

for k,v in trans_adj_list.items():
    print(k, ':', v)
```

Week 5.5 Countling Triplets

Problem Statement:
You are given an undirected, complete graph G that contains N vertices. Each edge is colored in either white or black. You are required to determine the number of triplets (i, j, k) (1 ≤ i < j < k ≤ N) of vertices such that the edges (i, j), (j, k), (i, k) are of the same color. There are M white edges and (N (N-1)/2) – M black edges.

Input format:

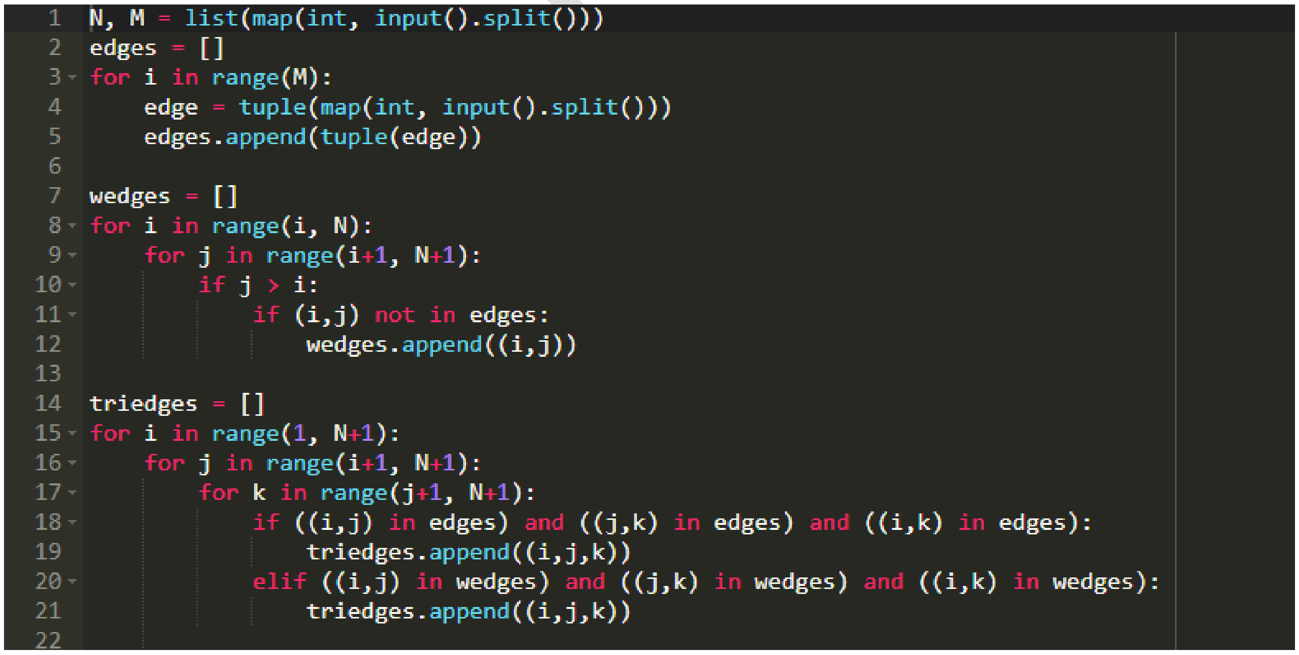
First line: Two integers – N and M (3 ≤ N ≤ 105 , 1 ≤ M ≤ 3 * 105)

(i+1)th line: Two integers – ui and vi (1 ≤ ui , vi ≤ N) denoting that the edge (ui , vi) is white in color.

Note: The conditions (ui , vi) ≠ (uj , vj) and (ui , vi) ≠ (vj, uj) are satisfied for all 1 ≤ i < j ≤ M.

Output format:

Print an integer that denotes the number of triples that satisfy the mentioned condition.



```python
N, M = list(map(int, input().split()))
edges = []
for i in range(M):
    edge = tuple(map(int, input().split()))
    edges.append(tuple(edge))

wedges = []
for i in range(i, N):
    for j in range(i+1, N+1):
        if j > i:
            if (i,j) not in edges:
                wedges.append((i,j))

triedges = []
for i in range(1, N+1):
    for j in range(i+1, N+1):
        for k in range(j+1, N+1):
            if ((i,j) in edges) and ((j,k) in edges) and ((i,k) in edges):
                triedges.append((i,j,k))
            elif ((i,j) in wedges) and ((j,k) in wedges) and ((i,k) in wedges):
                triedges.append((i,j,k))

```

Week 9 (2 Questions)

Week 9.1 Breadth First Search

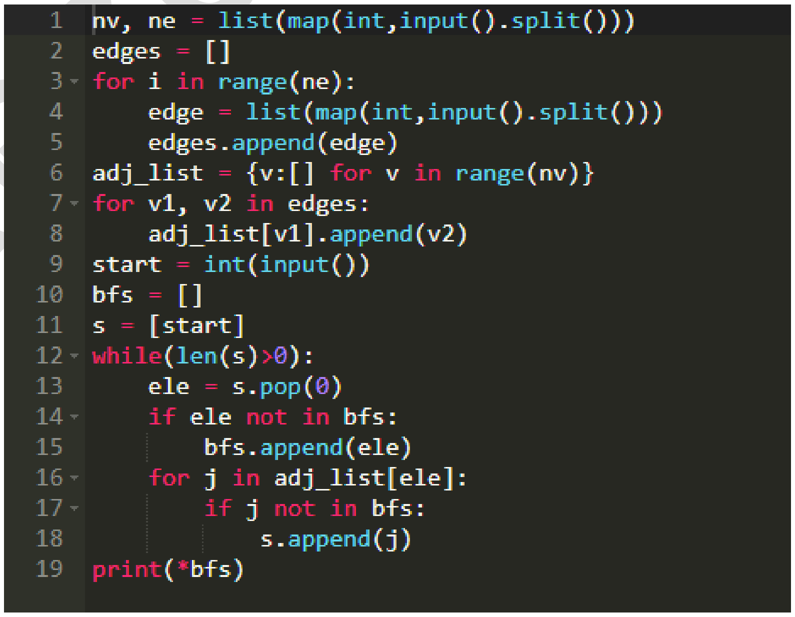
The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level. For a given graph G, print BFS traversal from a given source vertex

Input Format:

1. The first line contains two integers n and e, representing the number of vertices and edges in the graph respectively.
1. The next e lines each contain two integers u and v, representing an edge between vertex u and vertex v.
1. The last line contains an integer s, representing the source vertex from which the BFS traversal will start.

Output Format:

- A single line containing the vertices visited in the order of the BFS traversal starting from the source vertex.



```python
nv, ne = list(map(int,input().split()))
edges = []
for i in range(ne):
    edge = list(map(int,input().split()))
    edges.append(edge)
adj_list = {v:[] for v in range(nv)}
for v1, v2 in edges:
    adj_list[v1].append(v2)
start = int(input())
bfs = []
s = [start]
while(len(s)>0):
    ele = s.pop(0)
    if ele not in bfs:
        bfs.append(ele)
    for j in adj_list[ele]:
        if j not in bfs:
            s.append(j)
print(*bfs)
```

Week 9.2 Depth First Search

Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

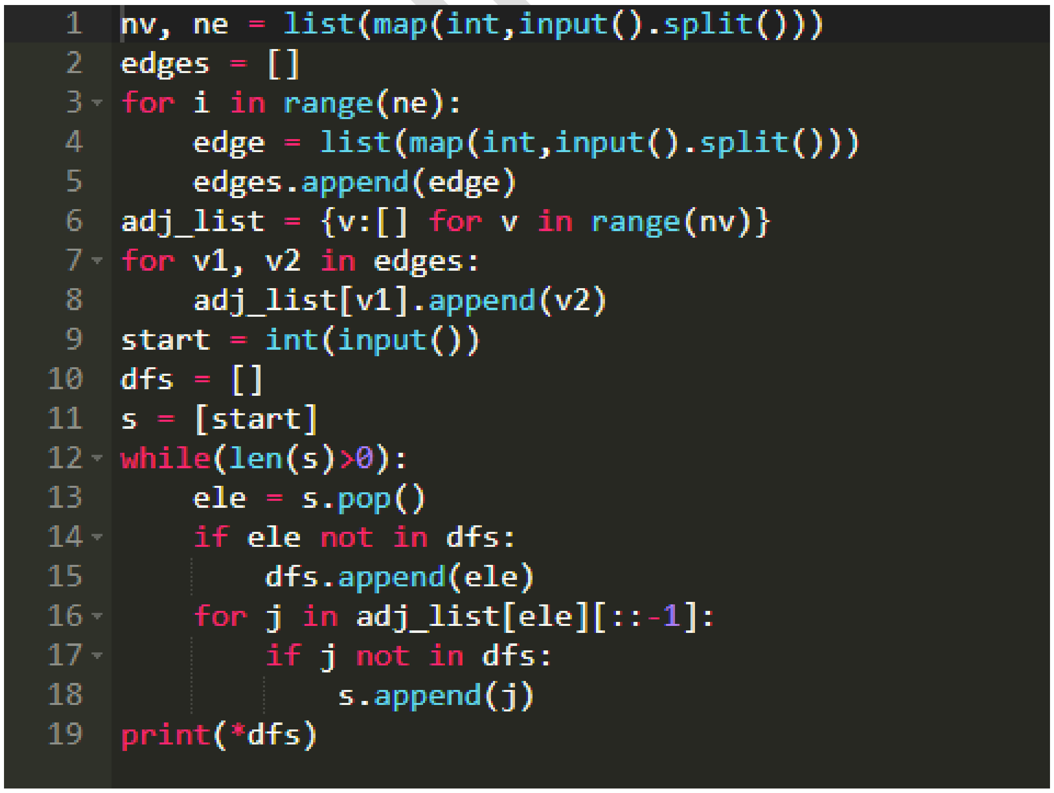
For a given graph G, print DFS traversal from a given source vertex.

Input Format:

1. The first line contains two integers n and e, representing the number of vertices and edges in the graph respectively.
2. The next e lines each contain two integers u and v, representing an edge between vertex u and vertex v.
3. The last line contains an integer s, representing the source vertex from which the DFS traversal will start.

Output Format:

- A single line containing the vertices visited in the order of the DFS traversal starting from the source vertex.



```python
nv, ne = list(map(int,input().split()))
edges = []
for i in range(ne):
    edge = list(map(int,input().split()))
    edges.append(edge)
adj_list = {v:[] for v in range(nv)}
for v1, v2 in edges:
    adj_list[v1].append(v2)
start = int(input())
dfs = []
s = [start]
while(len(s)>0):
    ele = s.pop()
    if ele not in dfs:
        dfs.append(ele)
    for j in adj_list[ele][::-1]:
        if j not in dfs:
            s.append(j)
print(*dfs)
```

Week 10 (2 Questions)

Week 10.1 Kruskal’s Algorithm

Problem Statement:

In Kruskal’s algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last.

MST using Kruskal’s algorithm:

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal’s algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Input:

6, 10

7, 0, 1

8, 0, 2

3, 1, 2

6, 1, 3

5, 1, 5

4, 2, 3

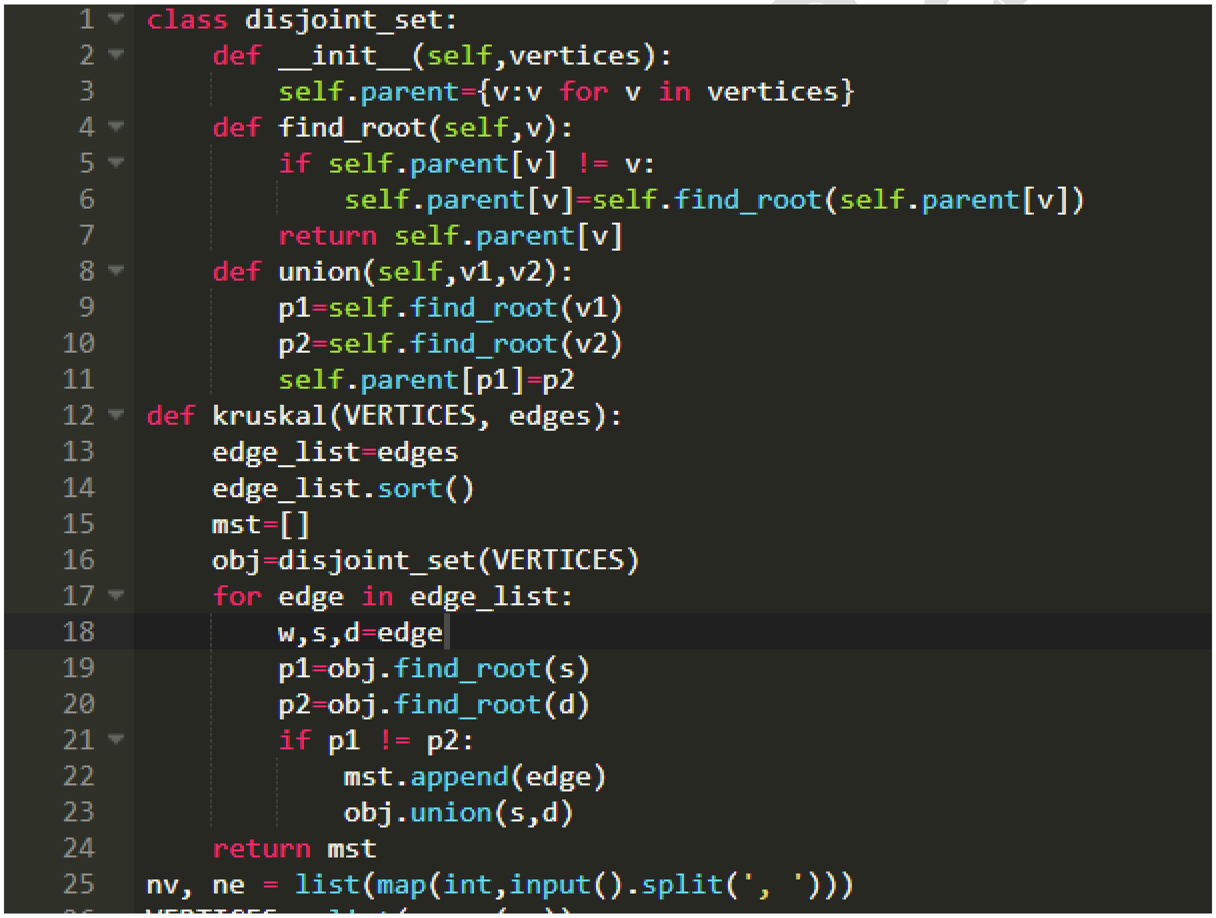
3, 2, 5

5, 3, 4

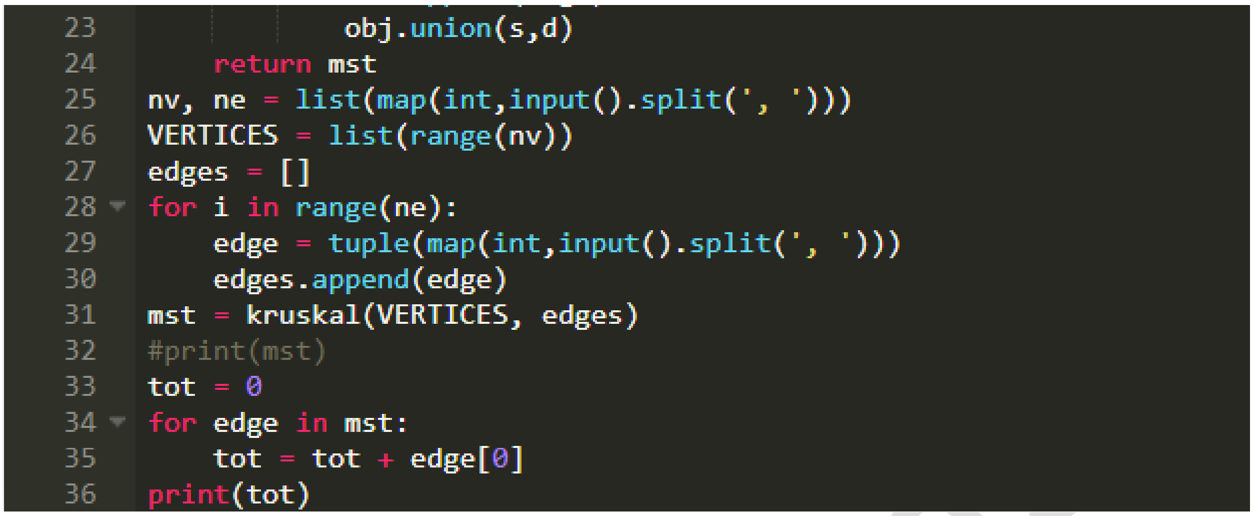
2, 3, 5

2, 4, 5

Output:17



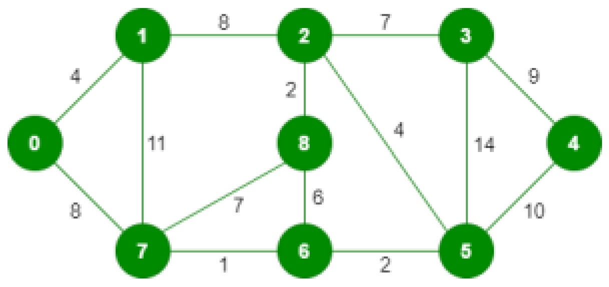
```python
class disjoint_set:
    def __init__(self,vertices):
        self.parent={v:v for v in vertices}
    def find_root(self,v):
        if self.parent[v] != v:
            self.parent[v]=self.find_root(self.parent[v])
        return self.parent[v]
    def union(self,v1,v2):
        p1=self.find_root(v1)
        p2=self.find_root(v2)
        self.parent[p1]=p2
def kruskal(VERTICES, edges):
    edge_list=edges
    edge_list.sort()
    mst=[]
    obj=disjoint_set(VERTICES)
    for edge in edge_list:
        w,s,d=edge
        p1=obj.find_root(s)
        p2=obj.find_root(d)
        if p1 != p2:
            mst.append(edge)
            obj.union(s,d)
    return mst
nv, ne = list(map(int,input().split(', ')))
```



```
            obj.union(s,d)
    return mst
nv, ne = list(map(int,input().split(', ')))
VERTICES = list(range(nv))
edges = []
for i in range(ne):
    edge = tuple(map(int,input().split(', ')))
    edges.append(edge)
mst = kruskal(VERTICES, edges)
#print(mst)
tot = 0
for edge in mst:
    tot = tot + edge[0]
print(tot)
```

Example:

For the given graph G find the minimum cost spanning tree.



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Week 10.2 Prim's Algorithm

Problem Statement:
The Prim's algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Prim's Algorithm:

he working of Prim's algorithm can be described by using the following steps:

1. Determine an arbitrary vertex as the starting vertex of the MST.

2. Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).

3. Find edges connecting any tree vertex with the fringe vertices.

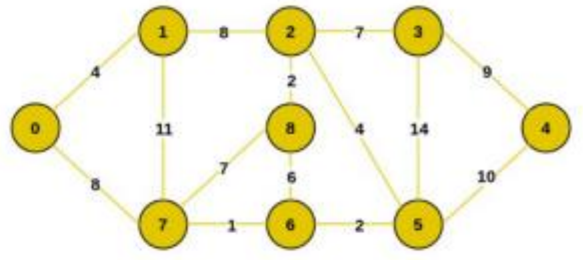
4. Find the minimum among these edges.

5. Add the chosen edge to the MST if it does not form any cycle.

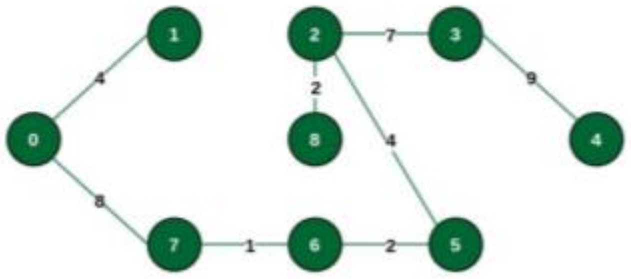
6. Return the MST and exit

Explanation:

Input: For the given graph G find the minimum cost spanning tree.



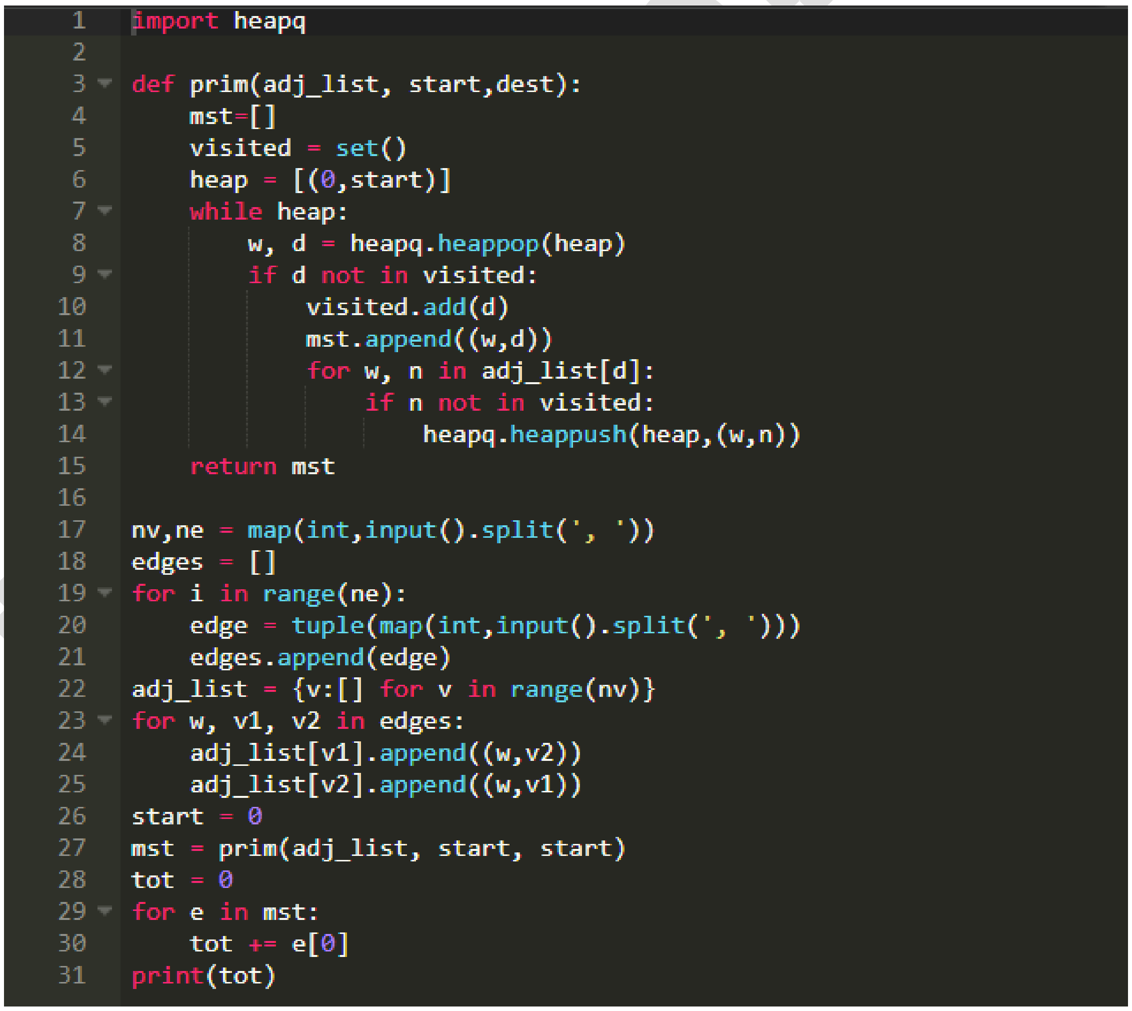
Output: The final structure of the MST is as follows and the weight of the edges of the MST is = 17



Output:

Edge Weight

Cost of spanning tree is : **17**



```
import heapq

def prim(adj_list, start,dest):
    mst=[]
    visited = set()
    heap = [(0,start)]
    while heap:
        w, d = heapq.heappop(heap)
        if d not in visited:
            visited.add(d)
            mst.append((w,d))
            for w, n in adj_list[d]:
                if n not in visited:
                    heapq.heappush(heap,(w,n))
    return mst

nv,ne = map(int,input().split(', '))
edges = []
for i in range(ne):
    edge = tuple(map(int,input().split(', ')))
    edges.append(edge)
adj_list = {v:[] for v in range(nv)}
for w, v1, v2 in edges:
    adj_list[v1].append((w,v2))
    adj_list[v2].append((w,v1))
start = 0
mst = prim(adj_list, start, start)
tot = 0
for e in mst:
    tot += e[0]
print(tot)
```

Week 11 (5 Questions)

Week 11.1 Bisection Method

The Bisection method is also called the interval halving method, the binary search method or the dichotomy method. This method is used to find root of an equation in a given interval that is value of 'x' for which $f(x) = 0$. The method is based on The Intermediate Value Theorem which states that if $f(x)$ is a continuous function and there are two real numbers a and b such that $f(a) * f(b)$ 0 and $f(b) < 0$), then it is guaranteed that it has at least one root between them.

Bisection Method Procedure

To solve bisection method problems, given below is the step-by-step explanation of the working of the bisection method algorithm for a given function $f(x)$:

Step 1: Choose two values, a and b such that $f(a) > 0$ and $f(b) < 0$.

Step 2: Calculate a midpoint c as the arithmetic mean between a and b such that $c = (a + b) / 2$. This is called interval halving.

Step 3: Evaluate the function f for the value of c.

Step 4: The root of the function is found only if the value of $f(c) = 0$.

Step 5: If $(c) \neq 0$, then we need to check the sign:

a. we replace a with c if f(c) has the same sign as f(a) and we keep the same value for b

b. we replace b with c if f(c) has the same sign as f(b), and we keep the same value for a

c. To get the right value with the new value of a or b, we go back to step 2 And recalculate

Input Format:

First line contains the coefficients of the equation

Second line contains the range where the root lies

Third line specifies the number of decimals to round the root of the given equation

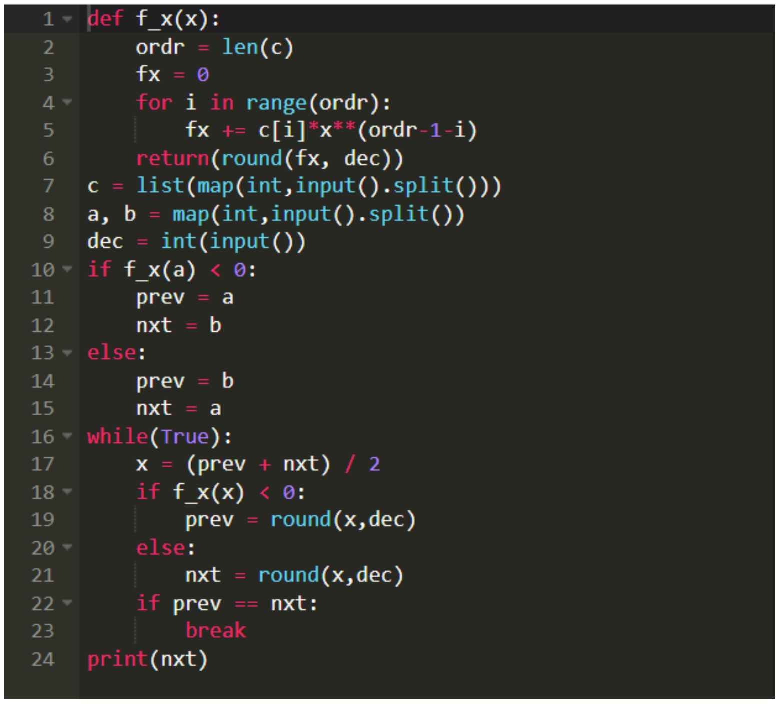
Output Format:

Output is root rounded to given number of decimals in the input

==================================

Sample test case:

Input:	**Output:**
2 0 -2 -5	1.601
1 2	
3	



```python
def f_x(x):
    ordr = len(c)
    fx = 0
    for i in range(ordr):
        fx += c[i]*x**(ordr-1-i)
    return(round(fx, dec))
c = list(map(int,input().split()))
a, b = map(int,input().split())
dec = int(input())
if f_x(a) < 0:
    prev = a
    nxt = b
else:
    prev = b
    nxt = a
while(True):
    x = (prev + nxt) / 2
    if f_x(x) < 0:
        prev = round(x,dec)
    else:
        nxt = round(x,dec)
    if prev == nxt:
        break
print(nxt)
```

Week 11.2 Method of False Position

Problem Statement:

Given a function f(x) on floating number x and two numbers 'a' and 'b' such that f(a)*f(b) < 0 and f(x) is continuous in [a, b]. Here f(x) represents algebraic or transcendental equation. Find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).

Input Format:

First line contains Algebraic equations coefficients a,b,c,d as list of elements (assuming $ax^3+bx^2 + c x + d = 0$)

Second line contains the boundary values [a,b]

Third line indicates how many decimal points to consider

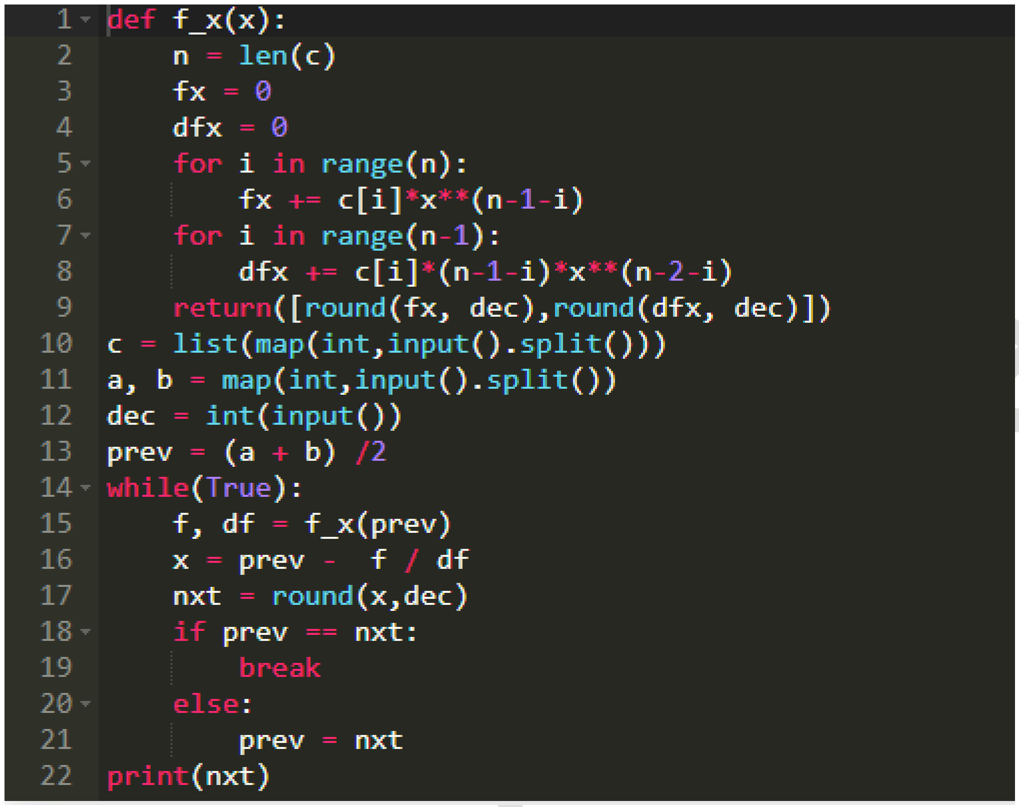
Output format:

Displays the root of the equation as output for the specified number of decimals

=====================================

Sample test case:

Input:	**output:**
1 0 -2 -5	2.0945
2 3	
4	



```python
def f_x(x):
    n = len(c)
    fx = 0
    dfx = 0
    for i in range(n):
        fx += c[i]*x**(n-1-i)
    for i in range(n-1):
        dfx += c[i]*(n-1-i)*x**(n-2-i)
    return([round(fx, dec),round(dfx, dec)])
c = list(map(int,input().split()))
a, b = map(int,input().split())
dec = int(input())
prev = (a + b) /2
while(True):
    f, df = f_x(prev)
    x = prev -  f / df
    nxt = round(x,dec)
    if prev == nxt:
        break
    else:
        prev = nxt
print(nxt)
```

11. 2 Method of False Position

Week 11.3 Newton Raphson Method

Given a function f(x) on floating number x and an initial guess for root, find root of function in interval. Here f(x) represents algebraic or transcendental equation.

Input: A function of x (for example x3 – x 2 + 2), derivative function of x (3x2 – 2x for above example) and an initial guess x0 = -20

Output: The value of root is: -1.00 or any other value close to root.

Algorithm:

Input: initial x, func(x), derivFunc(x)

Output: Root of Func()

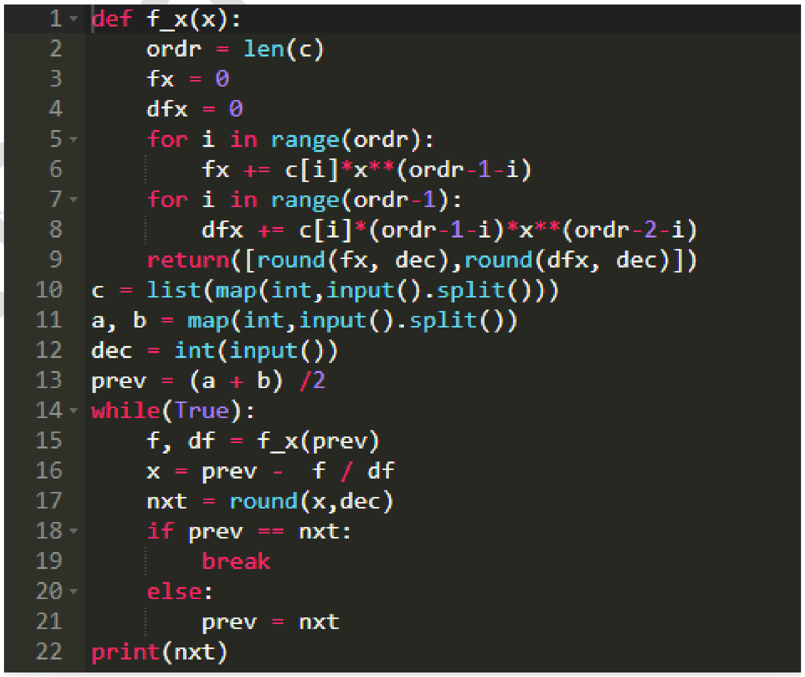
1. Compute values of func(x) and derivFunc(x) for given initial x

2. Compute h: h = func(x) / derivFunc(x)

3. While h is greater than allowed error ε

• h = func(x) / derivFunc(x)

• x = x – h



```python
def f_x(x):
    ordr = len(c)
    fx = 0
    dfx = 0
    for i in range(ordr):
        fx += c[i]*x**(ordr-1-i)
    for i in range(ordr-1):
        dfx += c[i]*(ordr-1-i)*x**(ordr-2-i)
    return([round(fx, dec),round(dfx, dec)])
c = list(map(int,input().split()))
a, b = map(int,input().split())
dec = int(input())
prev = (a + b) /2
while(True):
    f, df = f_x(prev)
    x = prev -  f / df
    nxt = round(x,dec)
    if prev == nxt:
        break
    else:
        prev = nxt
print(nxt)
```

Week 11.4 Secant Method

The secant method is used to find the root of an equation f(x) = 0. It is started from two distinct estimates x1 and x2 for the root. It is an iterative procedure involving linear interpolation to a root. The iteration stops if the difference between two intermediate values is less than the convergence factor.

Input Format:

First line contains Algebraic equations coefficients a,b,c,d as list of elements (assuming $ax^3+bx^2 + c x + d = 0$)

Second line contains the boundary values [a,b]

Third line indicates how many decimal points to consider

Output format:

Displays the root of the equation as output for the specified number of decimals

====================================

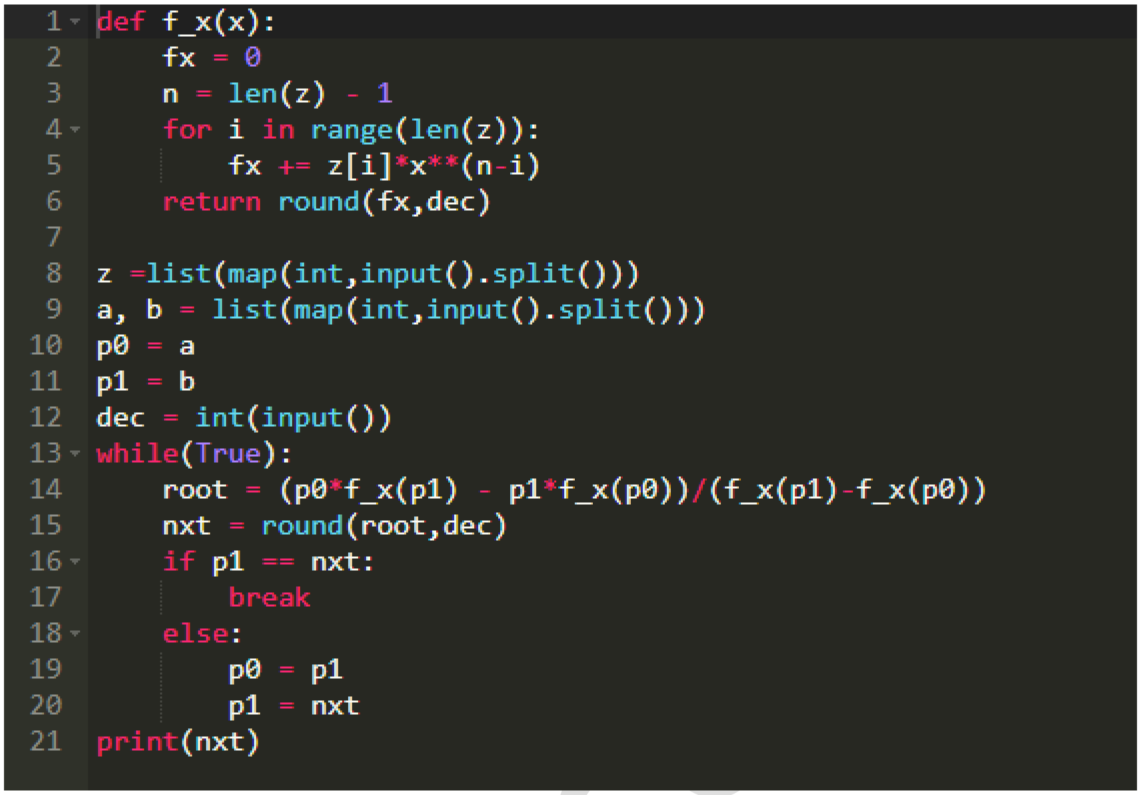
Sample test case:

Input:

```
1 0 -1 -1

1 2

5
```

Output

```
1.32472
```



```python
def f_x(x):
    fx = 0
    n = len(z) - 1
    for i in range(len(z)):
        fx += z[i]*x**(n-i)
    return round(fx,dec)

z =list(map(int,input().split()))
a, b = list(map(int,input().split()))
p0 = a
p1 = b
dec = int(input())
while(True):
    root = (p0*f_x(p1) - p1*f_x(p0))/(f_x(p1)-f_x(p0))
    nxt = round(root,dec)
    if p1 == nxt:
        break
    else:
        p0 = p1
        p1 = nxt
print(nxt)
```

11.4 Secant Method

Week 11.5 Muller Method

Given a function f(x) on floating number x and three initial distinct guesses for root of the function, find the root of function. Here, f(x) can be an algebraic or transcendental function.

Input: A function f(x) = x + 2x + 10x - 20 and three initial guesses - 0, 1 and 2 .

Output: The value of the root is 1.3688 or any other value within permittable deviation from the root.

Input: A function f(x) = x - 5x + 2 and three initial guesses - 0, 1 and 2.

Output: The value of the root is 0.4021 or any other value within permittable deviation from the root.

Input Format:

First line contains the coefficients of the equation

Second line contains the three initial values

Third line specifies the number of decimals to round the root of the given equation

Output Format:

Output is root rounded to given number of decimals in the input

=====================================

Sample test case:

Input:

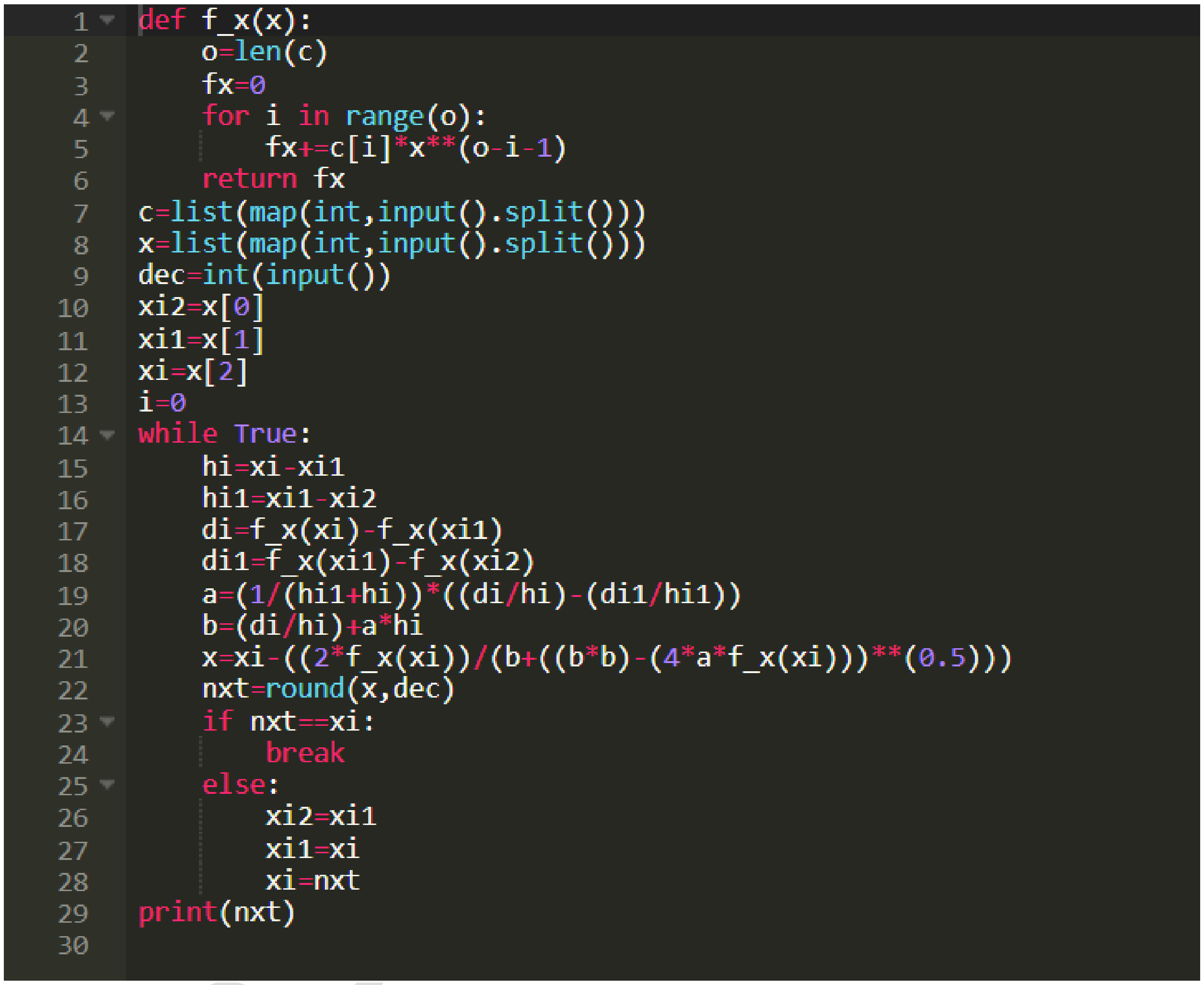
1 2 10 -20

0 1 2

4

Output:

1.3688



```
def f_x(x):
    o=len(c)
    fx=0
    for i in range(o):
        fx+=c[i]*x**(o-i-1)
    return fx
c=list(map(int,input().split()))
x=list(map(int,input().split()))
dec=int(input())
xi2=x[0]
xi1=x[1]
xi=x[2]
i=0
while True:
    hi=xi-xi1
    hi1=xi1-xi2
    di=f_x(xi)-f_x(xi1)
    di1=f_x(xi1)-f_x(xi2)
    a=(1/(hi1+hi))*((di/hi)-(di1/hi1))
    b=(di/hi)+a*hi
    x=xi-((2*f_x(xi))/(b+((b*b)-(4*a*f_x(xi)))**(0.5)))
    nxt=round(x,dec)
    if nxt==xi:
        break
    else:
        xi2=xi1
        xi1=xi
        xi=nxt
print(nxt)
```

Week 12 (3 Questions)

Week 12.1 Trapezoidal Rule for Approximate Value of Definite Integral

Trapezoidal rule is used to find the approximation of a definite integral. The basic idea in Trapezoidal rule is to assume the region under the graph of the given function to be a trapezoid and calculate its area.

$$\int_a^b f(x)\,dx \approx (b-a)\left[\frac{f(a)+f(b)}{2}\right]$$

Input Format:

First line contains the coefficients of the equation

Second line contains the integral interval [a, b]

Third line specifies the number of sub intervals n - integer

Output Format:

Output is area under the curve

Note: The function for algebraic equation rounded to 4 digits

=======================================

Sample test case:

Input:

2 0 -4 1

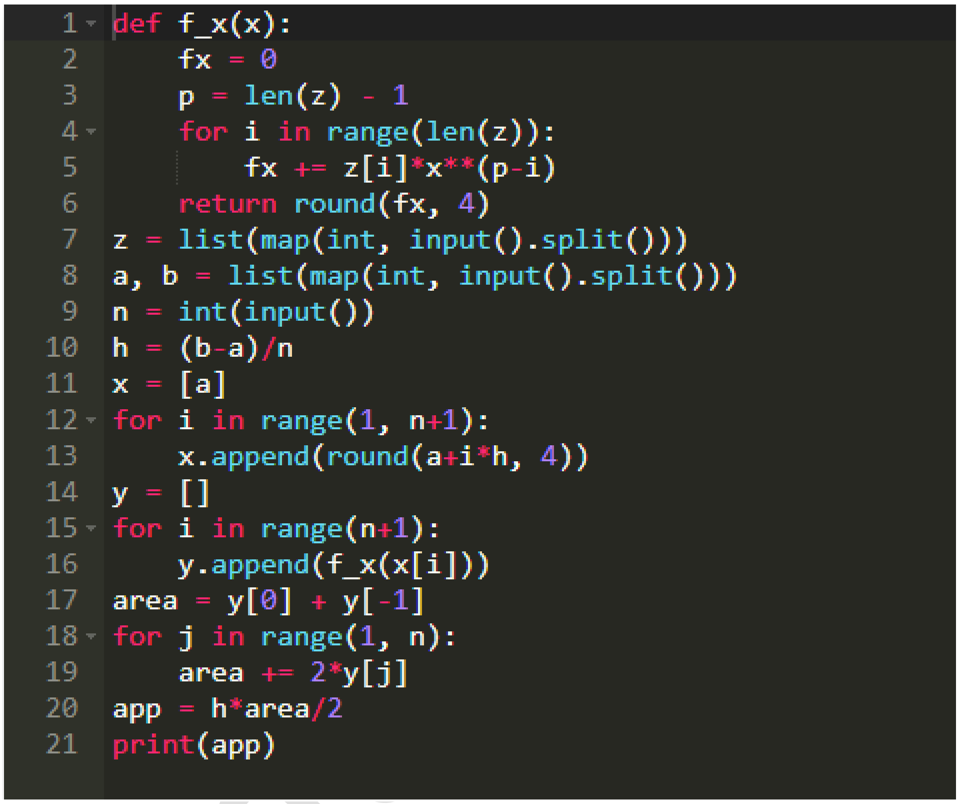
2 4

4

Output:

99.5

==



```python
def f_x(x):
    fx = 0
    p = len(z) - 1
    for i in range(len(z)):
        fx += z[i]*x**(p-i)
    return round(fx, 4)
z = list(map(int, input().split()))
a, b = list(map(int, input().split()))
n = int(input())
h = (b-a)/n
x = [a]
for i in range(1, n+1):
    x.append(round(a+i*h, 4))
y = []
for i in range(n+1):
    y.append(f_x(x[i]))
area = y[0] + y[-1]
for j in range(1, n):
    area += 2*y[j]
app = h*area/2
print(app)
```

Week 12.2 Simpson’s 1/3 Rule

Simpson’s 1/3 rule is a method for numerical approximation of definite integrals. Specifically, it is the following approximation:

$$\int_a^b f(x)\,dx = h/3\,[(y_0 + y_n) + 4(y_1 + y_3 + y_5 + \dots + y_{n-1}) + 2(y_2 + y_4 + y_6 + \dots + y_{n-2})]$$

Input Format:

First line contains the coefficients of the equation

Second line contains the integral interval [a, b]

Third line specifies the number of sub intervals n - integer

Output Format:

Output is area under the curve

Note: The function for algebraic equation rounded to 4 digits

=====================================

Sample test case:

Input:

1 0 0 0 10

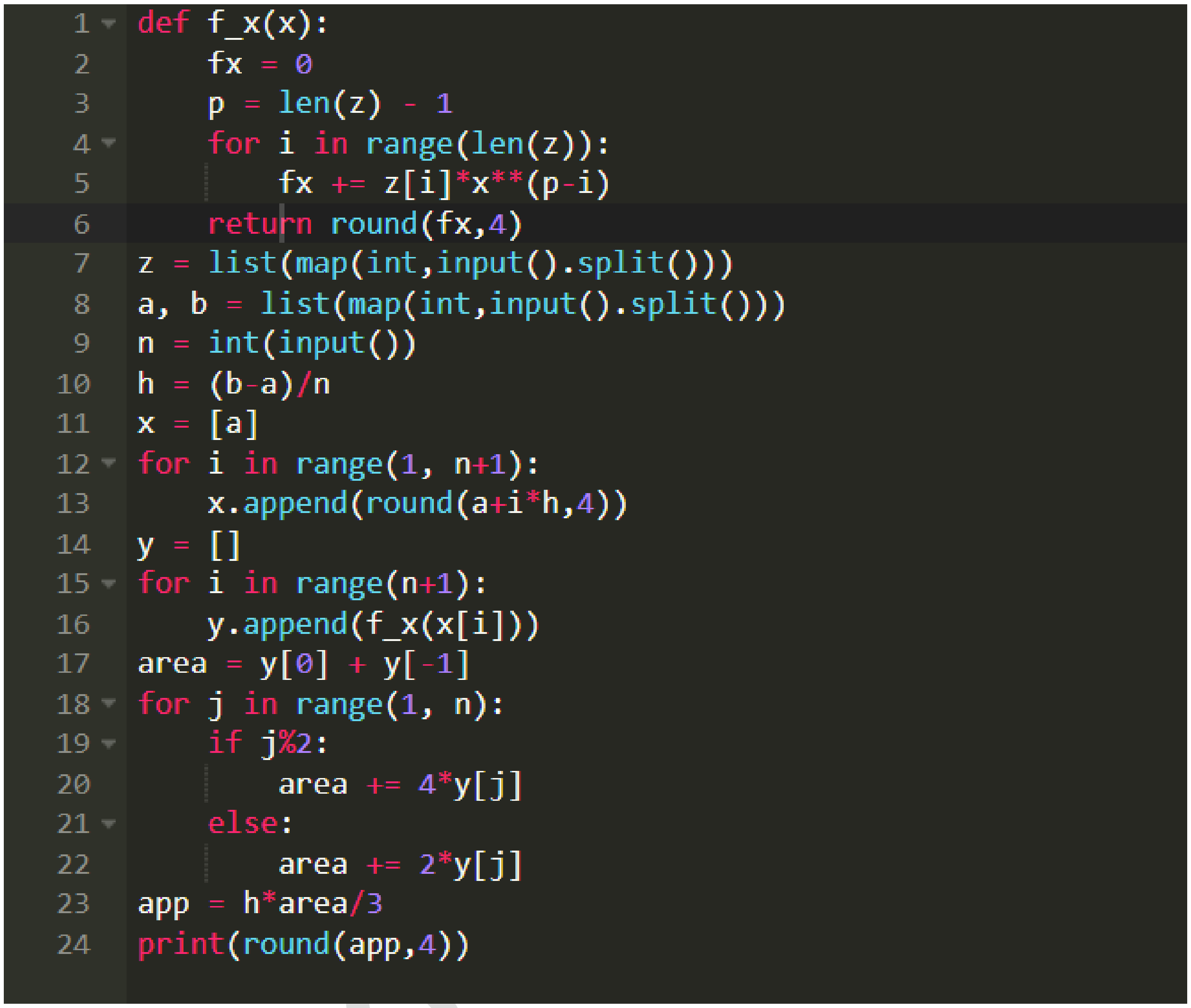
0 4

4

Output:

245.3333

=====================================



```python
def f_x(x):
    fx = 0
    p = len(z) - 1
    for i in range(len(z)):
        fx += z[i]*x**(p-i)
    return round(fx,4)
z = list(map(int,input().split()))
a, b = list(map(int,input().split()))
n = int(input())
h = (b-a)/n
x = [a]
for i in range(1, n+1):
    x.append(round(a+i*h,4))
y = []
for i in range(n+1):
    y.append(f_x(x[i]))
area = y[0] + y[-1]
for j in range(1, n):
    if j%2:
        area += 4*y[j]
    else:
        area += 2*y[j]
app = h*area/3
print(round(app,4))
```

12.2 Simpson's 1/3 Rule

Week 12.3 Simpson’s 3/8 Rule

The Simpson’s 3/8 rule was developed by Thomas Simpson. This method is used for performing numerical integrations. This method is generally used for numerical approximation of definite integrals. Here, parabolas are used to approximate each part of curve.

Input Format:

First line contains the coefficients of the equation

Second line contains the integral interval [a, b]

Third line specifies the number of sub intervals n - integer

Output Format:

Output is area under the curve

Note: The function for algebraic equation rounded to 4 digits

==

Sample test case:

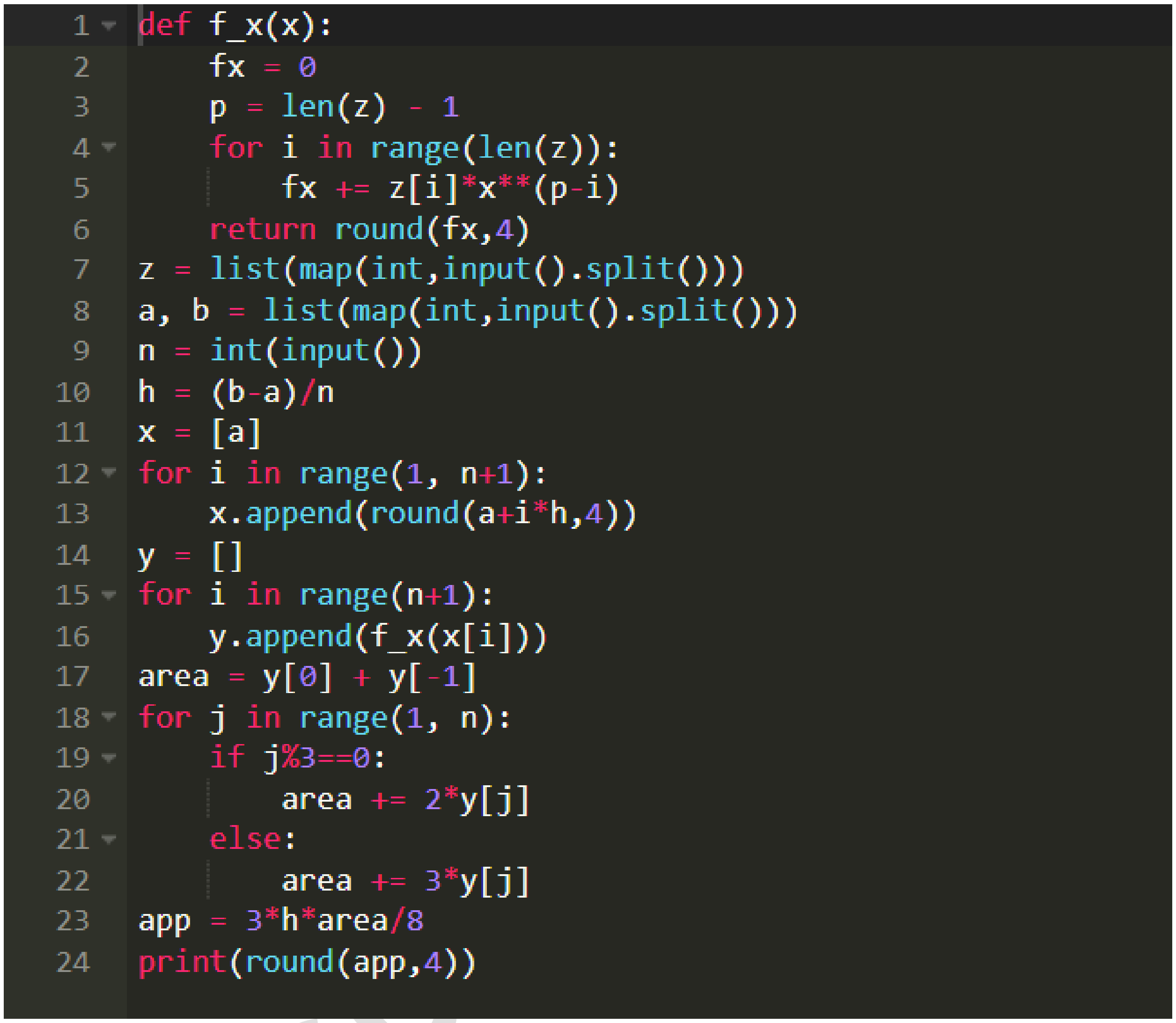
Input:

```
1 0 0 0 10

0 4

4
```

Output:

```
213.375
```

===



```python
def f_x(x):
    fx = 0
    p = len(z) - 1
    for i in range(len(z)):
        fx += z[i]*x**(p-i)
    return round(fx,4)
z = list(map(int,input().split()))
a, b = list(map(int,input().split()))
n = int(input())
h = (b-a)/n
x = [a]
for i in range(1, n+1):
    x.append(round(a+i*h,4))
y = []
for i in range(n+1):
    y.append(f_x(x[i]))
area = y[0] + y[-1]
for j in range(1, n):
    if j%3==0:
        area += 2*y[j]
    else:
        area += 3*y[j]
app = 3*h*area/8
print(round(app,4))
```

12.3 Simpson's 3/8 Rule