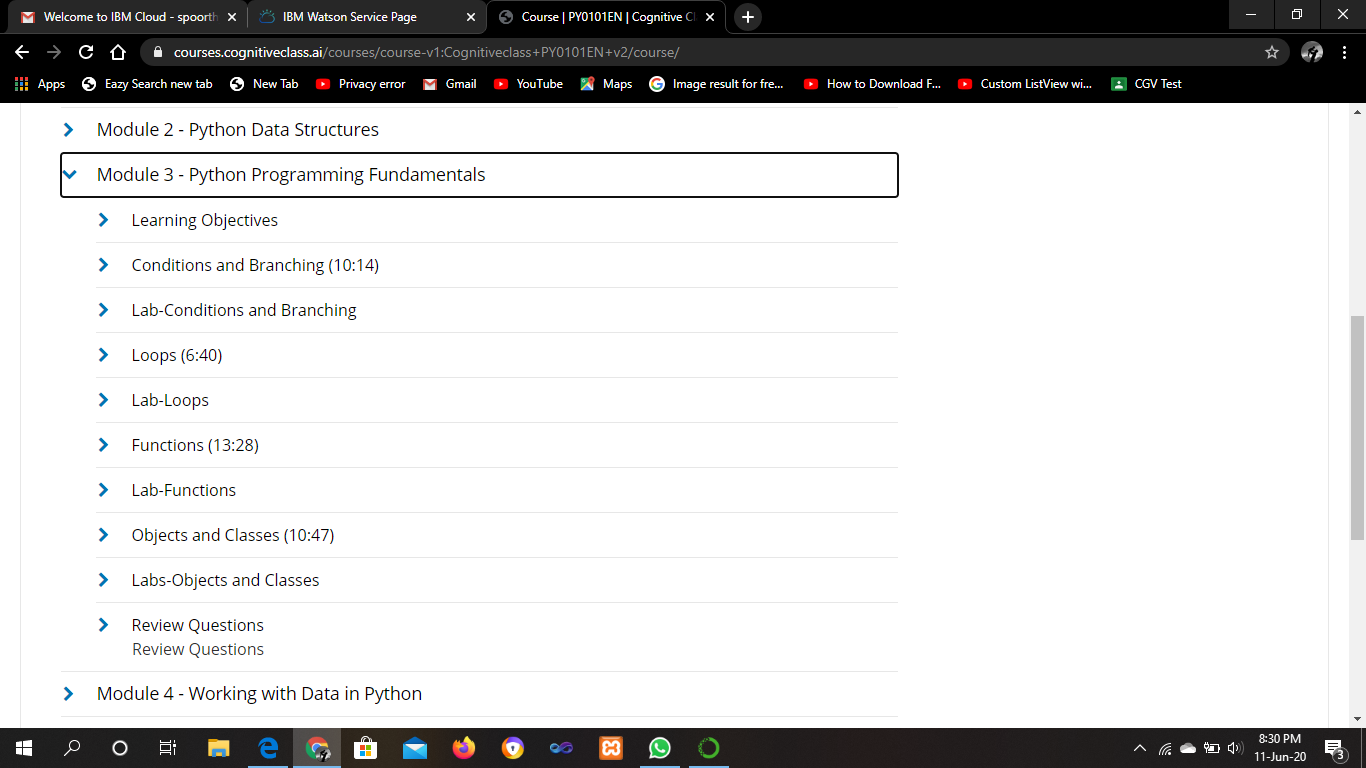
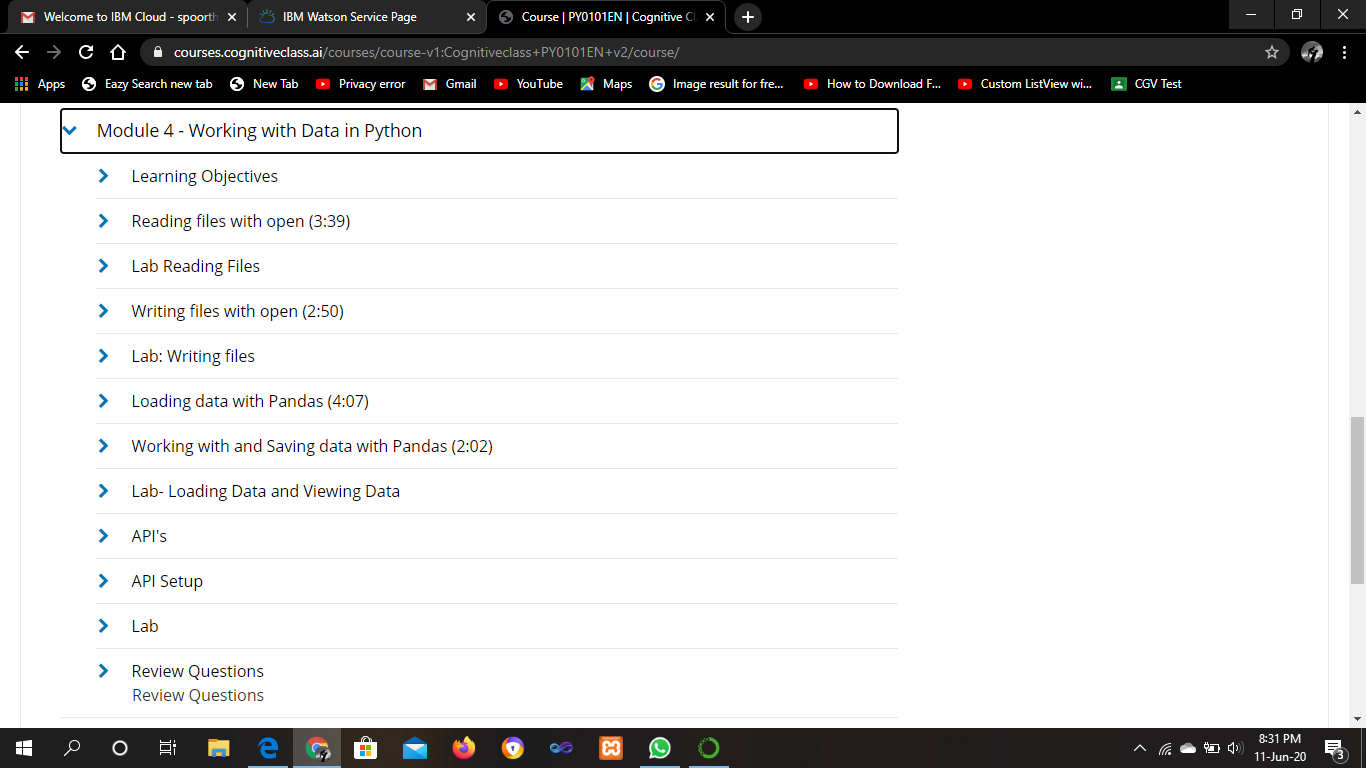
**DAILY ONLINE ACTIVITIES SUMMARY**

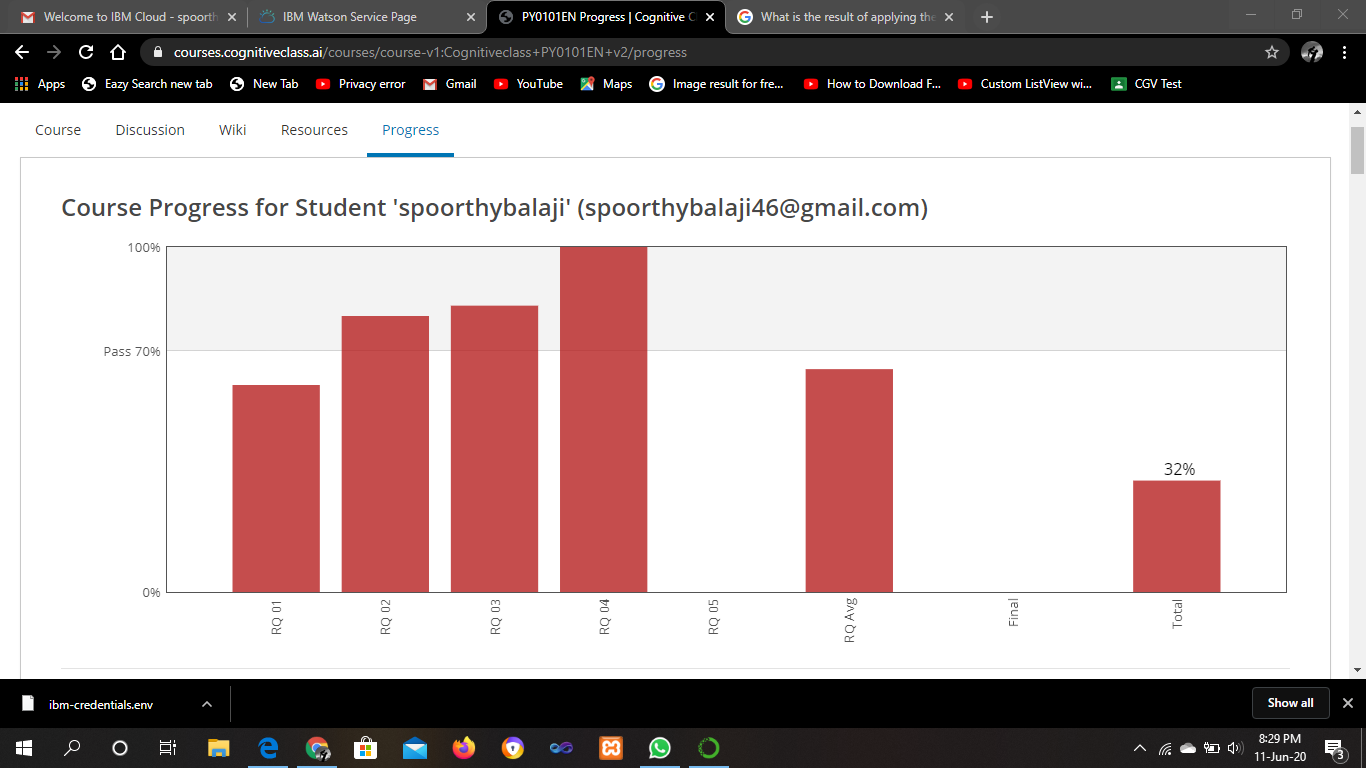
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Date:** | **12/06/2020** | | | | | **Name:** | **Prajna** | |
| **Sem & Sec** | **6th & A** | | | | | **USN:** | **4al17cs059** | |
| **Online Test Summary** | | | | | | | | |
| **Subject** | |  | | | | | | |
| **Max. Marks** | |  | | **Score** | | |  | |
| **Certification Course Summary** | | | | | | | | |
| **Course** | **Python for Data Science** | | | | | | | |
| **Certificate Provider** | | | **COGNITIVE CLASS.ia** | | **Duration** | | | **9hours** |
| **Coding Challenges** | | | | | | | | |
| **Problem Statement:** 4 Programs | | | | | | | | |
| **Status: Solved** | | | | | | | | |
| **Uploaded the report in Github** | | | | | **yes** | | | |
| **If yes Repository name** | | | | | <https://github.com/prajna-nayak-098/Daily-Report> | | | |
| **Uploaded the report in slack** | | | | | **yes** | | | |

IA TEST

CERTIFICATION COURSE







ONLINE CODING

**1. Python Program to find the average of all Items in a Dictionary Step1: Get the name as key and marks as value for n students Step2: find the average of the marks of all the students and print it**

dict={}

n=int(input("Enter the number of entries:"))

for i in range(0,n):

key = input("Enter the name: ")

value = int(input("Enter the marks: "))

dict[key] = value

print(dict)

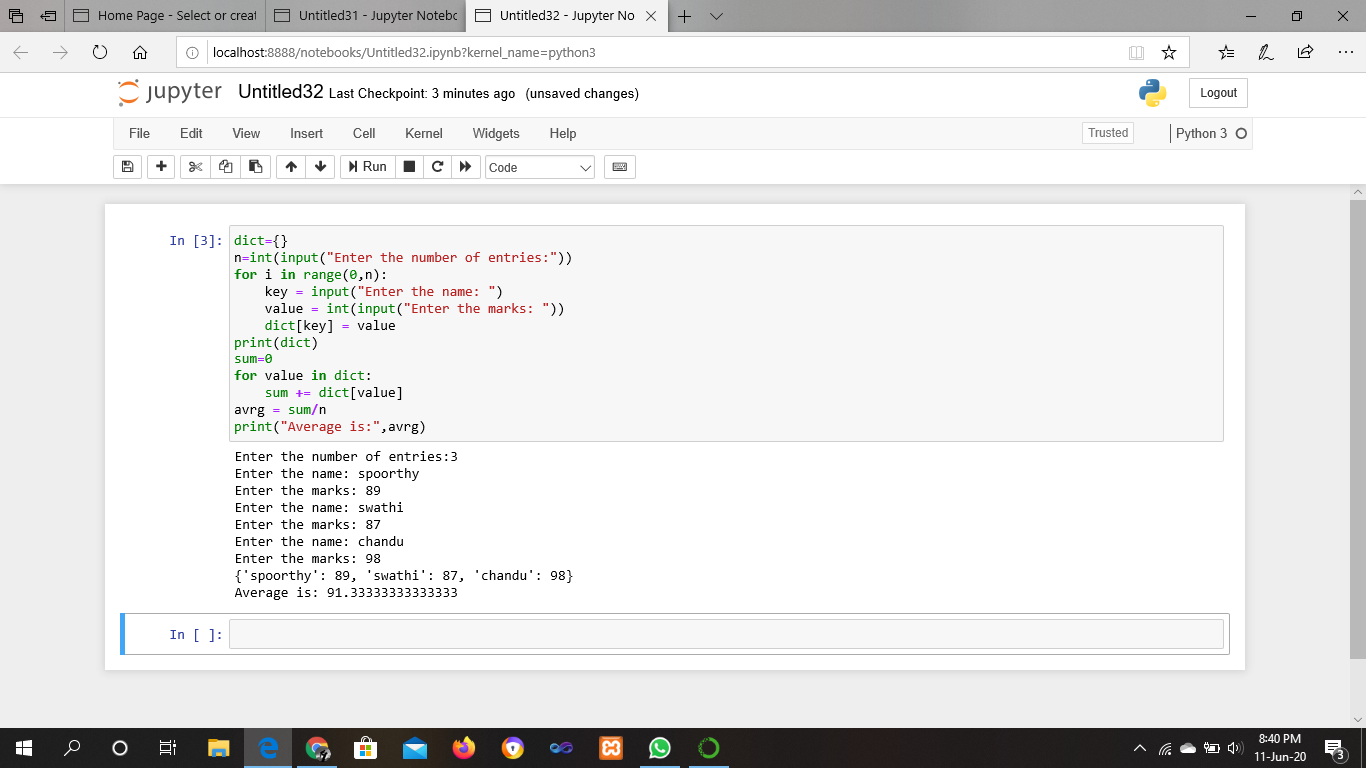
sum=0

for value in dict:

sum += dict[value]

avrg = sum/n

print("Average is:",avrg)



**2. Write a Java program to find the nodes which are at the maximum distance in a Binary Tree**

**import** java.util.ArrayList;

**public** **class** MaxDistance {

**public** **static** **class** Node{

**int** data;

        Node left;

        Node right;

**public** Node(**int** data){

**this**.data = data;

**this**.left = **null**;

**this**.right = **null**;

            }

          }

**public** Node root;

**int**[] treeArray;

**int** index = 0;

**public** MaxDistance(){

        root = **null**;

    }

**public** **int** calculateSize(Node node)

    {

**int** size = 0;

**if** (node == **null**)

**return** 0;

**else** {

            size = calculateSize (node.left) + calculateSize (node.right) + 1;

**return** size;

        }

    }

**public** **void** convertBTtoArray(Node node) {

**if**(root == **null**){

            System.out.println("Tree is empty");

**return**;

        }

**else** {

**if**(node.left != **null**)

                convertBTtoArray(node.left);

                        treeArray[index] = node.data;

            index++;

**if**(node.right != **null**)

                convertBTtoArray(node.right);

            }

    }

**public** **int** getDistance(Node temp, **int** n1) {

**if** (temp != **null**) {

**int** x = 0;

**if** ((temp.data == n1) || (x = getDistance(temp.left, n1)) > 0

                    || (x = getDistance(temp.right, n1)) > 0) {

**return** x + 1;

            }

**return** 0;

        }

**return** 0;

    }

**public** Node lowestCommonAncestor(Node temp, **int** node1, **int** node2) {

**if** (temp != **null**) {

**if** (temp.data == node1 || temp.data == node2) {

**return** temp;

            }

                   Node left = lowestCommonAncestor(temp.left, node1, node2);

            Node right = lowestCommonAncestor(temp.right, node1, node2);

**if** (left != **null** && right != **null**) {

**return** temp;

            }

**if** (left != **null**) {

**return** left;

            }

**if** (right != **null**) {

**return** right;

            }

        }

**return** **null**;

    }

**public** **int** findDistance(**int** node1, **int** node2) {

**int** d1 = getDistance(root, node1) - 1;

**int** d2 = getDistance(root, node2) - 1;

        Node ancestor = lowestCommonAncestor(root, node1, node2);

**int** d3 = getDistance(root, ancestor.data) - 1;

**return** (d1 + d2) - 2 \* d3;

    }

**public** **void** nodesAtMaxDistance(Node node) {

**int** maxDistance = 0, distance = 0;

        ArrayList<Integer> arr = **new** ArrayList<>();

**int** treeSize = calculateSize(node);

        treeArray = **new** **int**[treeSize];

        convertBTtoArray(node);

**for**(**int** i = 0; i < treeArray.length; i++) {

**for**(**int** j = i; j < treeArray.length; j++) {

                distance = findDistance(treeArray[i], treeArray[j]);

**if**(distance > maxDistance) {

                    maxDistance = distance;

                    arr.clear();

                    arr.add(treeArray[i]);

                    arr.add(treeArray[j]);

           }

**else** **if**(distance == maxDistance) {

                    arr.add(treeArray[i]);

                    arr.add(treeArray[j]);

                }

            }

        }

        System.out.println("Nodes which are at maximum distance: ");

**for**(**int** i = 0; i < arr.size(); i = i + 2) {

            System.out.println("( " + arr.get(i) + "," + arr.get(i+1) + " )");

        }

    }

**public** **static** **void** main(String[] args) {

        MaxDistance bt = **new** MaxDistance();

        bt.root = **new** Node(1);

        bt.root.left = **new** Node(2);

        bt.root.right = **new** Node(3);

        bt.root.left.left = **new** Node(4);

        bt.root.left.right = **new** Node(5);

        bt.root.right.left = **new** Node(6);

        bt.root.right.right = **new** Node(7);

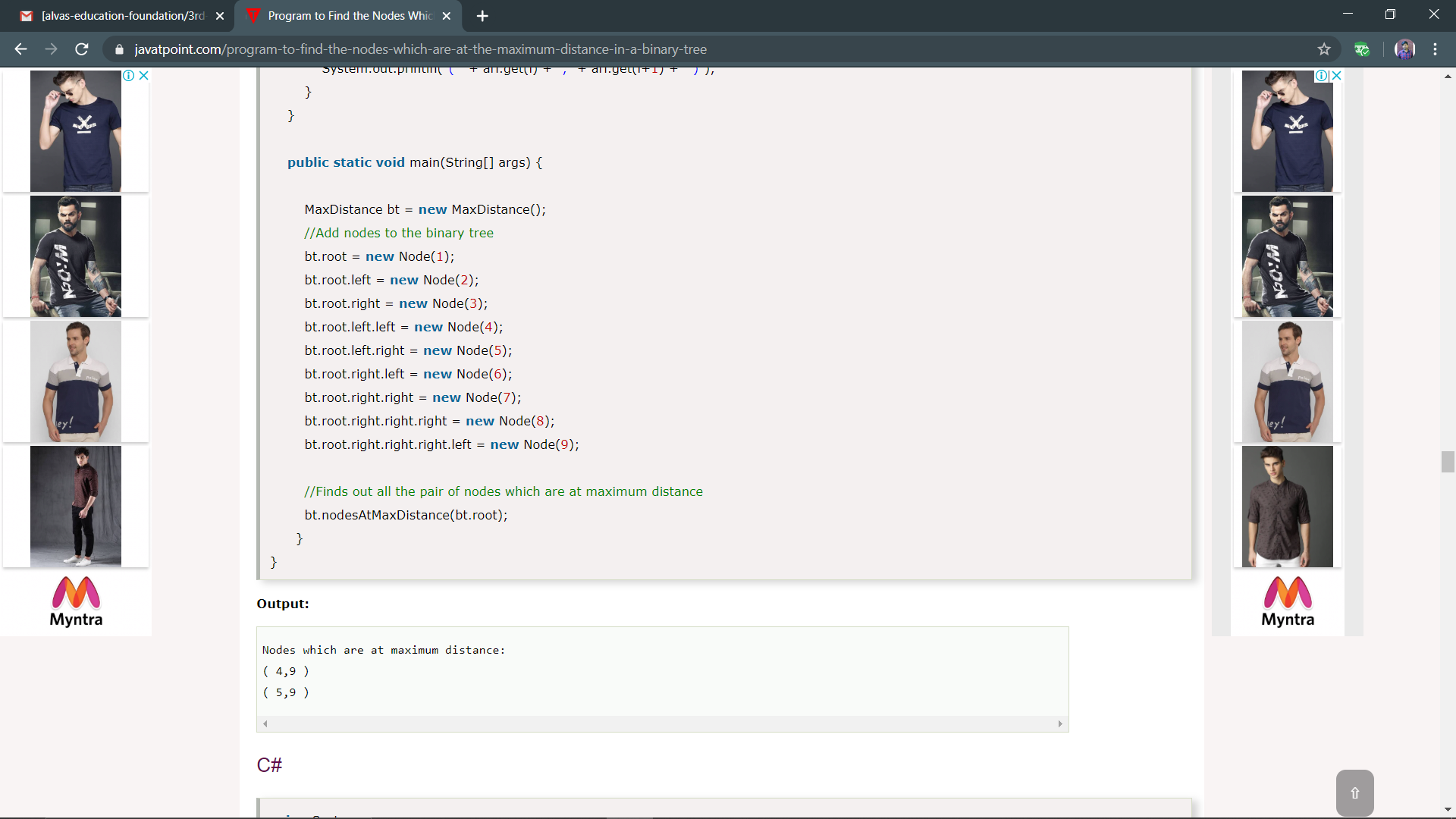
        bt.root.right.right.right = **new** Node(8);

        bt.root.right.right.right.left = **new** Node(9);

        bt.nodesAtMaxDistance(bt.root);

      }

}



3. Write a python function that converts a string to all uppercase, provided it contains at least 2 uppercase characters in the first 4 characters. Else print the string as it is

def to\_uppercase(str1):

    num\_upper = 0

    for letter in str1[:4]:

        if letter.upper() == letter:

            num\_upper += 1

    if num\_upper >= 2:

        return str1.upper()

    return str1

s=input("enter string: ")

print("Output: ",to\_uppercase(s))

