DAILY ONLINE ACTIVITIES SUMMARY

Date:	**11 June 2020**	**Name:**	**Asha Rudrappa Totagi**
Sem& Sec	**6thsem& A sec**	**USN:**	**4AL17CS015**

Online Test Summary

Subject	**Python Application And Programming**		
Max. Marks	**20**	**Score**	**18**

Certification Course Summary

Course	**Ethical Hacking**		
Certificate Provider	**Udemy**	**Duration**	**3 hours**

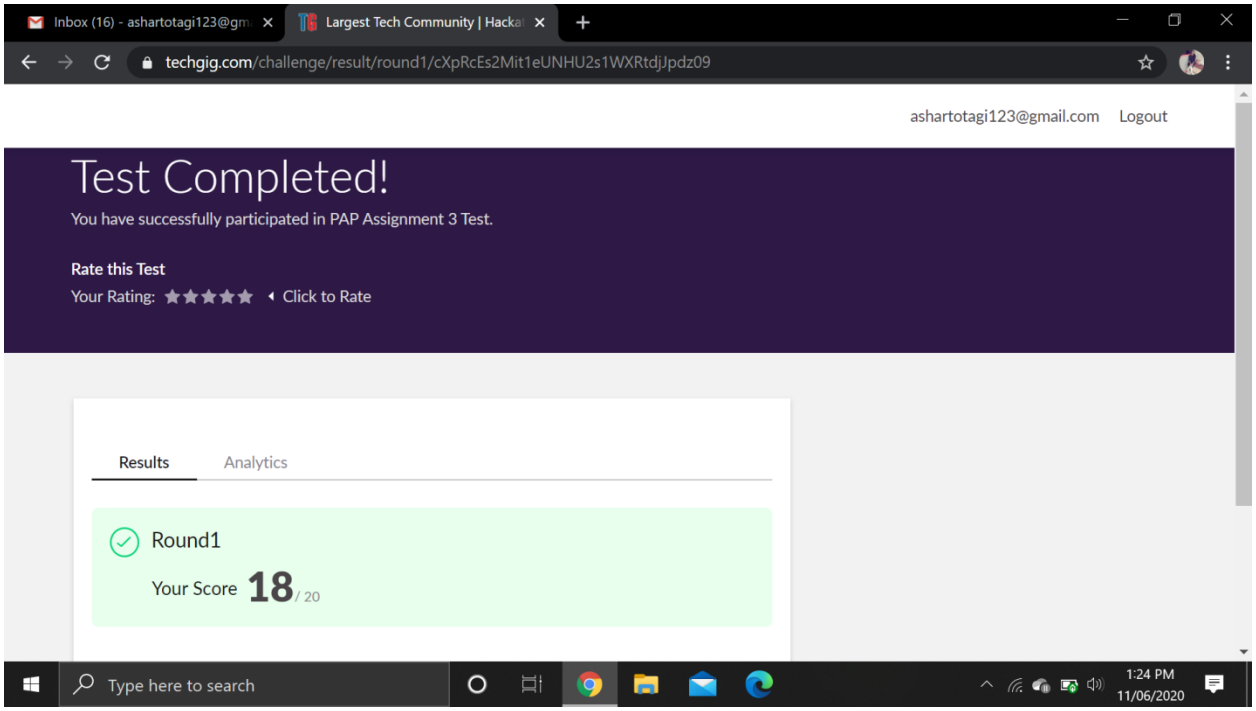
Coding Challenges

Problem Statement
Program 1: Write a Java program to find the nodes which are at the maximum distance in a Binary Tree

Program 2: Python program to print a given string in uppercase.

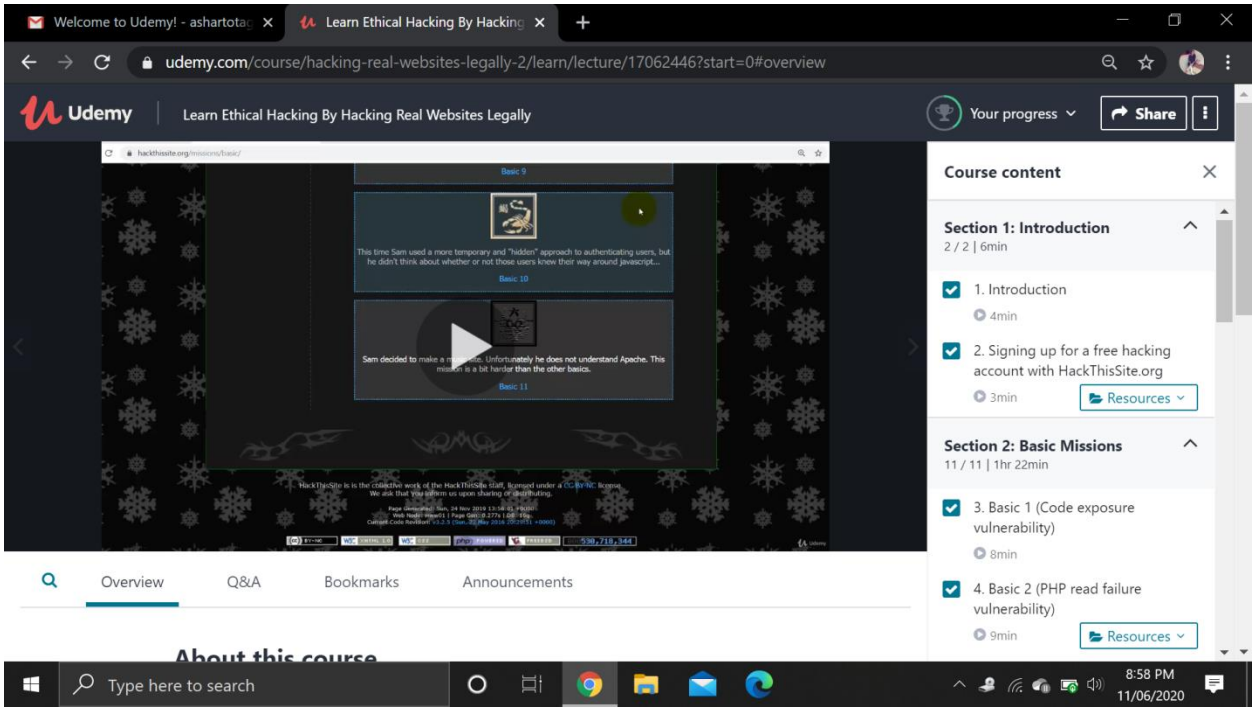
Status: DONE	
Uploaded the report in Github	**YES**
If yes Repository name	**Daily Status**
Uploaded the report in slack	**YES**

Online Test Details: (Attach the snapshot and briefly write the report for the same)



PAP Assignment Test 3 was held today i.e, 11 June 2020. Out of 20 marks I scored 18.

Certification Course Details: (Attach the snapshot and briefly write the report for the same



DAY 1 (11-06-2020) - Introduction to ethical hacking and basic missions to hack.

Coding Challenges Details: (Attach the snapshot and briefly write the report for the same)

Program 1:

```
import java.util.ArrayList;

public class MaxDistance {

//Represent a node of binary tree
public static class Node{
int data;
Node left;
      Node right;

      public Node(int data){
         //Assign data to the new node, set left and right children to null
         this.data = data;
         this.left = null;
         this.right = null;
         }
       }

   //Represent the root of binary tree
   public Node root;

   int[] treeArray;
   int index = 0;

   public MaxDistance(){
      root = null;
   }

   //calculateSize() will calculate size of tree
   public int calculateSize(Node node)
   {
      int size = 0;
      if (node == null)
       return 0;
      else {
         size = calculateSize (node.left) + calculateSize (node.right) + 1;
         return size;
      }
   }

   //convertBTtoArray() will convert binary tree to its array representation
```

```
    public void convertBTtoArray(Node node) {
        //Check whether tree is empty
        if(root == null){
            System.out.println("Tree is empty");
            return;
        }
        else {
            if(node.left != null)
                convertBTtoArray(node.left);
            //Adds nodes of binary tree to treeArray
            treeArray[index] = node.data;
            index++;
            if(node.right != null)
                convertBTtoArray(node.right);
            }
    }

    //getDistance() will find distance between root and a specific node
    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) {
                //x will store the count of number of edges between temp and node n1
                return x + 1;
            }
            return 0;
        }
        return 0;
    }

    //lowestCommonAncestor() will find out the lowest common ancestor for nodes node1 and no
de2
    public Node lowestCommonAncestor(Node temp, int node1, int node2) {
        if (temp != null) {
            //If root is equal to either of node node1 or node2, return root
            if (temp.data == node1 || temp.data == node2) {
                return temp;
            }

            //Traverse through left and right subtree
            Node left = lowestCommonAncestor(temp.left, node1, node2);
            Node right = lowestCommonAncestor(temp.right, node1, node2);

            //If node temp has one node(node1 or node2) as left child and one node(node1 or node2)
as right child
```

```
            //Then, return node temp  as lowest common ancestor
            if (left != null && right != null) {
               return temp;
            }

            //If nodes node1 and node2 are in left subtree
            if (left != null) {
               return left;
            }
            //If nodes node1 and node2 are in right subtree
            if (right != null) {
               return right;
            }
         }
         return null;
      }

      //findDistance() will find distance between two given nodes
      public int findDistance(int node1, int node2) {
         //Calculates distance of first node from root
         int d1 = getDistance(root, node1) - 1;
         //Calculates distance of second node from root
         int d2 = getDistance(root, node2) - 1;

         //Calculates lowest common ancestor of both the nodes
         Node ancestor = lowestCommonAncestor(root, node1, node2);

         //If lowest common ancestor is other than root then, subtract 2 * (distance of root to ancesto
r)
         int d3 = getDistance(root, ancestor.data) - 1;
         return (d1 + d2) - 2 * d3;
      }

      //nodesAtMaxDistance() will display the nodes which are at maximum distance
      public void nodesAtMaxDistance(Node node) {
         int maxDistance = 0, distance = 0;
         ArrayList<Integer> arr = new ArrayList<>();

         //Initialize treeArray
         int treeSize = calculateSize(node);
         treeArray = new int[treeSize];

         //Convert binary tree to its array representation
         convertBTtoArray(node);
```

```
        //Calculates distance between all the nodes present in binary tree and stores maximum dista
nce in variable maxDistance
        for(int i = 0; i < treeArray.length; i++) {
           for(int j = i; j < treeArray.length; j++) {
              distance = findDistance(treeArray[i], treeArray[j]);
              //If distance is greater than maxDistance then, maxDistance will hold the value of dista
nce
              if(distance > maxDistance) {
                 maxDistance = distance;
                 arr.clear();
                 //Add nodes at position i and j to treeArray
                 arr.add(treeArray[i]);
                 arr.add(treeArray[j]);
              }
              //If more than one pair of nodes are at maxDistance then, add all pairs to treeArray
              else if(distance == maxDistance) {
                 arr.add(treeArray[i]);
                 arr.add(treeArray[j]);
              }
           }
        }
        //Display all pair of nodes which are at maximum distance
        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();
        //Add nodes to the binary tree
        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);

        //Finds out all the pair of nodes which are at maximum distance
        bt.nodesAtMaxDistance(bt.root);
      }
}
```

Program 2:

```
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
string=input('Enter string:')
print(to_uppercase(string))
```