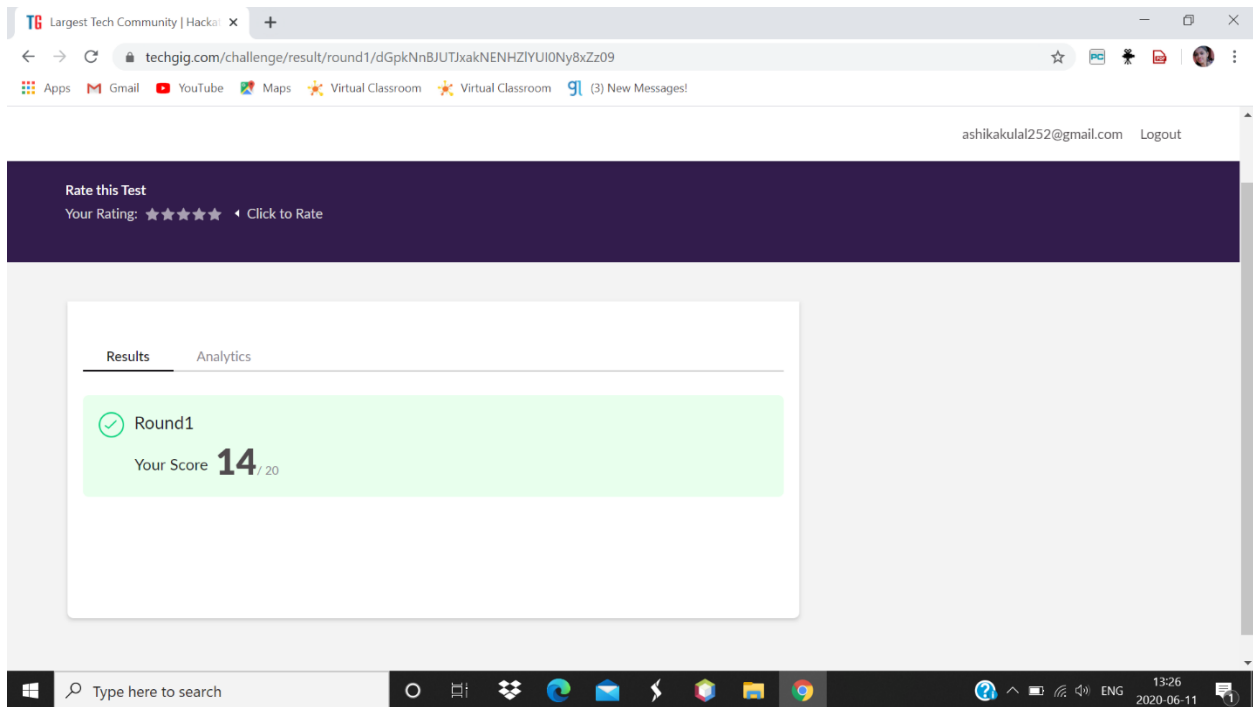


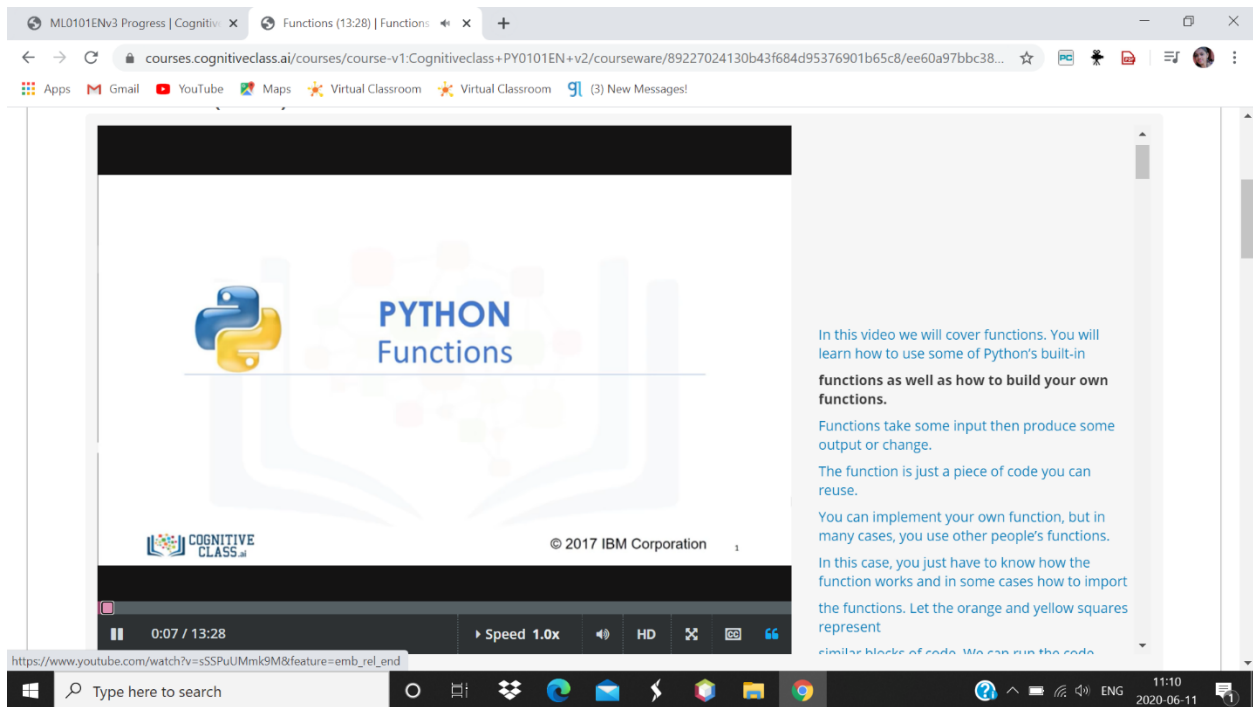
DAILY ONLINE ACTIVITIES SUMMARY

Date:	11-06-2020	Name:	ASHIKA
Sem & Sec	6 A	USN:	4AL17CS016
Online Test Summary			
Subject	PAP -ASSIGNMENT TEST3		
Max. Marks	20	Score	14
Certification Course Summary			
Course	Python for data science		
Certificate Provider	Cognitive class	Duration	5 hour
Coding Challenges			
Problem Statement: 1. Write a Java program to find the nodes which are at the maximum distance in a Binary Tree 2. Write a Python function to convert a given string to all uppercase if it contains at least 2 uppercase characters in the first 4 characters			
Status: done(executed)			
Uploaded the report in Github		yes	
If yes Repository name		https://github.com/ASHIKA-05/DAILY-REPORT	
Uploaded the report in slack		yes	

SUBJECT: PAP



CERTIFICATION COURSE



ONLINE CODEING

1. Write a Java program to find the nodes which are at the maximum distance in a Binary Tree
Description:

Algorithm

Define Node class which has three attributes namely: data left and right. Here, left represents the left child of the node and right represents the right child of the node. When a node is created, data will pass to data attribute of the node and both left and right will be set to null.

Define another class which has two attribute root and treeArray.

Root represents the root node of the tree and initializes it to null.

treeArray will store the array representation of the binary tree.

nodesAtMaxDistance() will find out the nodes which are present at the maximum distance:

It will calculate the distance between all the nodes present in the binary tree and store it in the variable distance.

MaxDistance keeps tracks of maximum possible distance between nodes. If maxDistance is less than distance, then the value of distance will be stored in maxDistance. Clears the array to get rid of previously stored values. Nodes which are at the maximum distance will be stored in an array arr.

If more than one pair of nodes is at maxDistance then, store them in the array arr.

a. calculateSize() will count the number of nodes present in the tree.

b. convertBTToArray() will convert the binary tree to its array representation by traversing the tree and adding elements to treeArray.

c. getDistance() will calculate the distance of a given node from the root.

d. LowestCommonAncestor() will find out the lowest common ancestor for the nodes n1 and n2.

If any of the nodes is equal to the root node, then return root as the lowest common ancestor.

Else, search nodes n1 and n2 in left subtree and right subtree.

If a node is found such that n1 is the left child of that node and n2 is right child of that node or vice-versa. Returns that node as the lowest common ancestor.

a. FindDistance() will calculate the distance between two nodes.

First, it calculates the distance of each node from the root node.

Subtract the 2 * distance of lowest common ancestor from this root node

```
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);

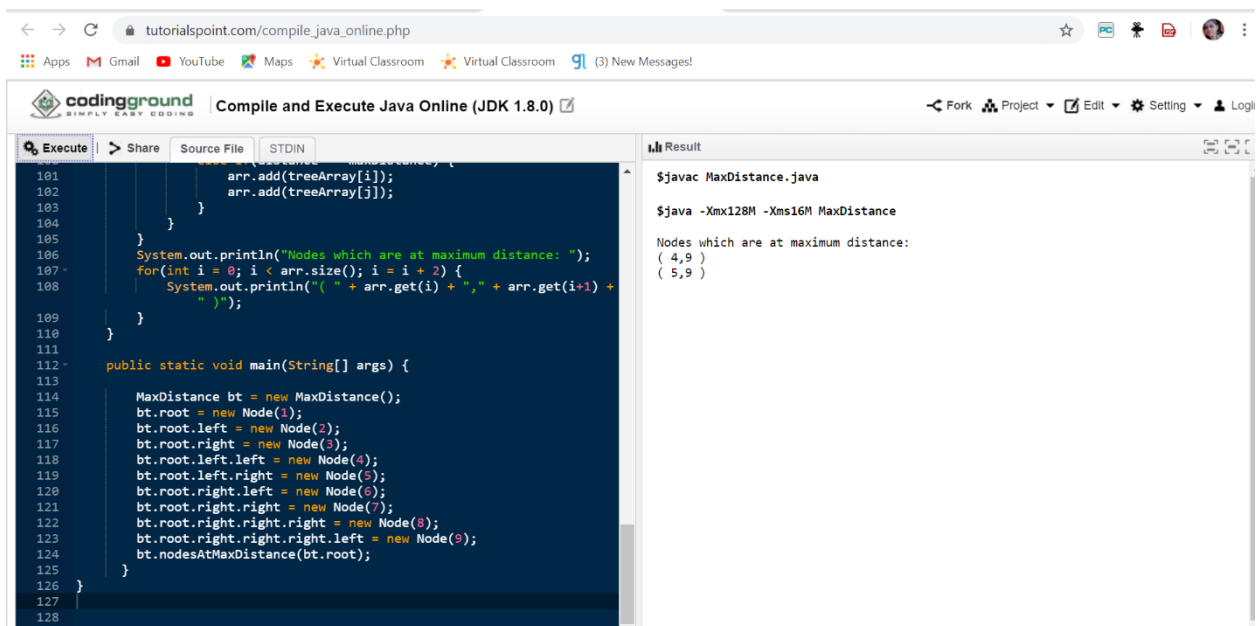
        bt.nodesAtMaxDistance(bt.root);

    }

}

```

Output:



The screenshot shows a web browser window with the URL `tutorialspoint.com/compile_java_online.php`. The page title is "Compile and Execute Java Online (JDK 1.8.0)". The code editor shows the following Java code:

```

101         arr.add(treeArray[i]);
102         arr.add(treeArray[j]);
103     }
104 }
105 }
106 System.out.println("Nodes which are at maximum distance: ");
107 for(int i = 0; i < arr.size(); i = i + 2) {
108     System.out.println(" ( " + arr.get(i) + ", " + arr.get(i+1) +
109         " )");
110 }
111 }
112 public static void main(String[] args) {
113     MaxDistance bt = new MaxDistance();
114     bt.root = new Node(1);
115     bt.root.left = new Node(2);
116     bt.root.right = new Node(3);
117     bt.root.left.left = new Node(4);
118     bt.root.left.right = new Node(5);
119     bt.root.right.left = new Node(6);
120     bt.root.right.right = new Node(7);
121     bt.root.right.right.right = new Node(8);
122     bt.root.right.right.right.left = new Node(9);
123     bt.nodesAtMaxDistance(bt.root);
124 }
125 }
126 }
127 }
128 }

```

The output window shows the following commands and results:

```

$javac MaxDistance.java

$java -Xmx128M -Xms16M MaxDistance

Nodes which are at maximum distance:
( 4,9 )
( 5,9 )

```

2. Write a Python function to convert a given string to all uppercase if it contains at least 2 uppercase characters in the first 4 characters

```

def to_uppercase(str1):
    num_upper = 0

    for letter in str1[:4]:
        if letter.upper() == letter:

```

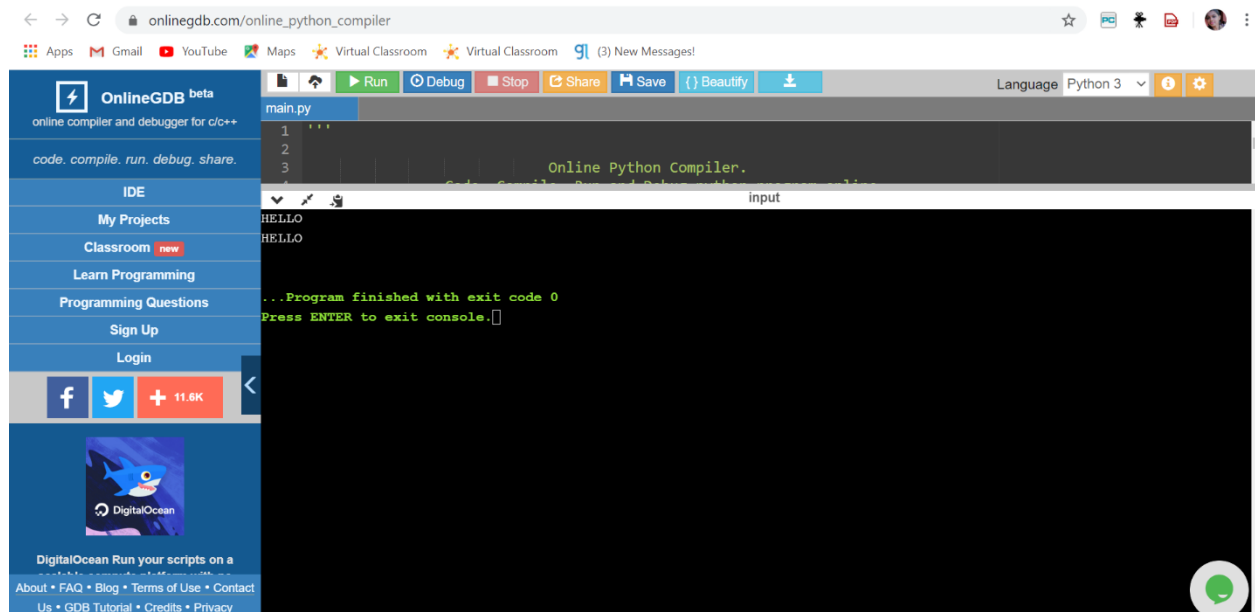
```
        num_upper += 1

    if num_upper >= 2:
        return str1.upper()

    return str1


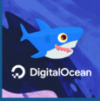
print(to_uppercase('Python'))

print(to_uppercase('PyThon'))
```



← → ↻ onlinegdb.com/online_python_compiler ☆ 📄 🖨️ 📧

📱 Apps 📧 Gmail 📺 YouTube 🗺️ Maps 🌟 Virtual Classroom 🌟 Virtual Classroom 📧 (3) New Messages!

**OnlineGDB** beta
online compiler and debugger for c/c++
code. compile. run. debug. share.
IDE
My Projects
Classroom **new**
Learn Programming
Programming Questions
Sign Up
Login
f 🐦 + 11.6K

DigitalOcean Run your scripts on a
About • FAQ • Blog • Terms of Use • Contact
Us • GDB Tutorial • Credits • Privacy
© 2016 - 2020 GDB Online

main.py

```
1
2
3
```

Run Debug Stop Share Save {} Beautify

Language Python 3 ⌵ ⚙️

```
Online Python Compiler.
Hello
Hello
...Program finished with exit code 0
Press ENTER to exit console.
```

input

local keyword

help

