**DAILY ONLINE ACTIVITIES SUMMARY**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Date:** | **12-06-2020** | | | | | **Name:** | **Anvitha U** | |
| **Sem & Sec** | **6th&A** | | | | | **USN:** | **4AL17CS009** | |
| **Online Test Summary** | | | | | | | | |
| **Subject** | | **No exam** | | | | | | |
| **Max. Marks** | | **-** | | **Score** | | | **-** | |
| **Certification Course Summary** | | | | | | | | |
| **Course** | **INTRODUCTION TO FULL STACK DEVELOPMENT.** | | | | | | | |
| **Certificate Provider** | | | Greatlearning  Academy | | **Duration** | | | 20hours |
| **Coding Challenges** | | | | | | | | |
| **Problem Statement:**1 Write a Python program to implement Magic Square.  2. Python program to print the pattern. 3. Write a Java program to find maximum width of a binary tree. | | | | | | | | |
| **Status: Done** | | | | | | | | |
| **Uploaded the report in Github** | | | | | **YES** | | | |
| **If yes Repository name** | | | | | <https://github.com/anvithauppoor/online_coding_activity> | | | |
| **Uploaded the report in slack** | | | | | **YES** | | | |

Online Test Details:

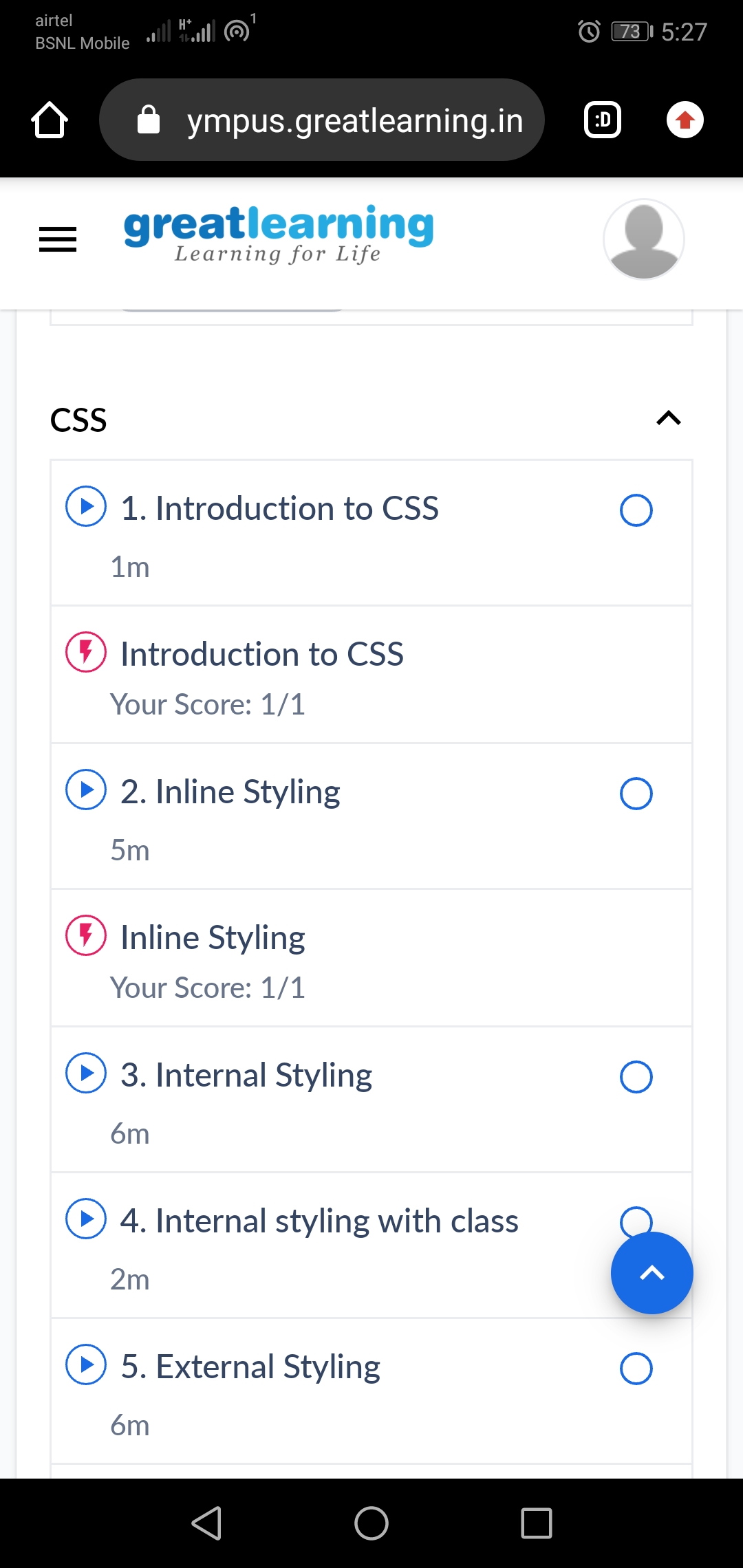
Subject:- **No exam.**

Certification Course Details:

**INTRODUCTION TO FULL STACK DEVELOPMENT.**

Today I have studied :

* Introduction to css.
* Introduction to css quiz.
* Inline styling.
* Inline styling quiz.
* Internet styling.
* Internal styling with class.
* External styling.



Coding Challenges Details:

1.Write a Python program to implement Magic Square

A magic square of order n is an arrangement of n^2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. A magic square contains the integers from 1 to n^2.

**The constant sum in every row, column and diagonal is called the magic constant or magic sum, M. The magic constant of a normal magic square depends only on n and has the following value:  
M = n(n^2+1)/2  
example  
Magic Square of size 5**

9 3 22 16 15  
2 21 20 14 8  
25 19 13 7 1  
18 12 6 5 24  
11 10 4 23 17  
Sum in each row & each column = 5\*(5^2+1)/2 = 65

def generateSquare(n):

magicSquare = [[0 for x in range(n)]

for y in range(n)]

i = n / 2

j = n - 1

num = 1

while num <= (n \* n):

if i == -1 and j == n:

j = n - 2

i = 0

else:

if j == n:

j = 0

if i < 0:

i = n - 1

if magicSquare[int(i)][int(j)]:

j = j - 2

i = i + 1

continue

else:

magicSquare[int(i)][int(j)] = num

num = num + 1

j = j + 1

i = i - 1

print ("Magic Square for n =", n)

print ("Sum of each row or column",n \* (n \* n + 1) / 2, "\n")

for i in range(0, n):

for j in range(0, n):

print('%2d ' % (magicSquare[i][j]),end = '')

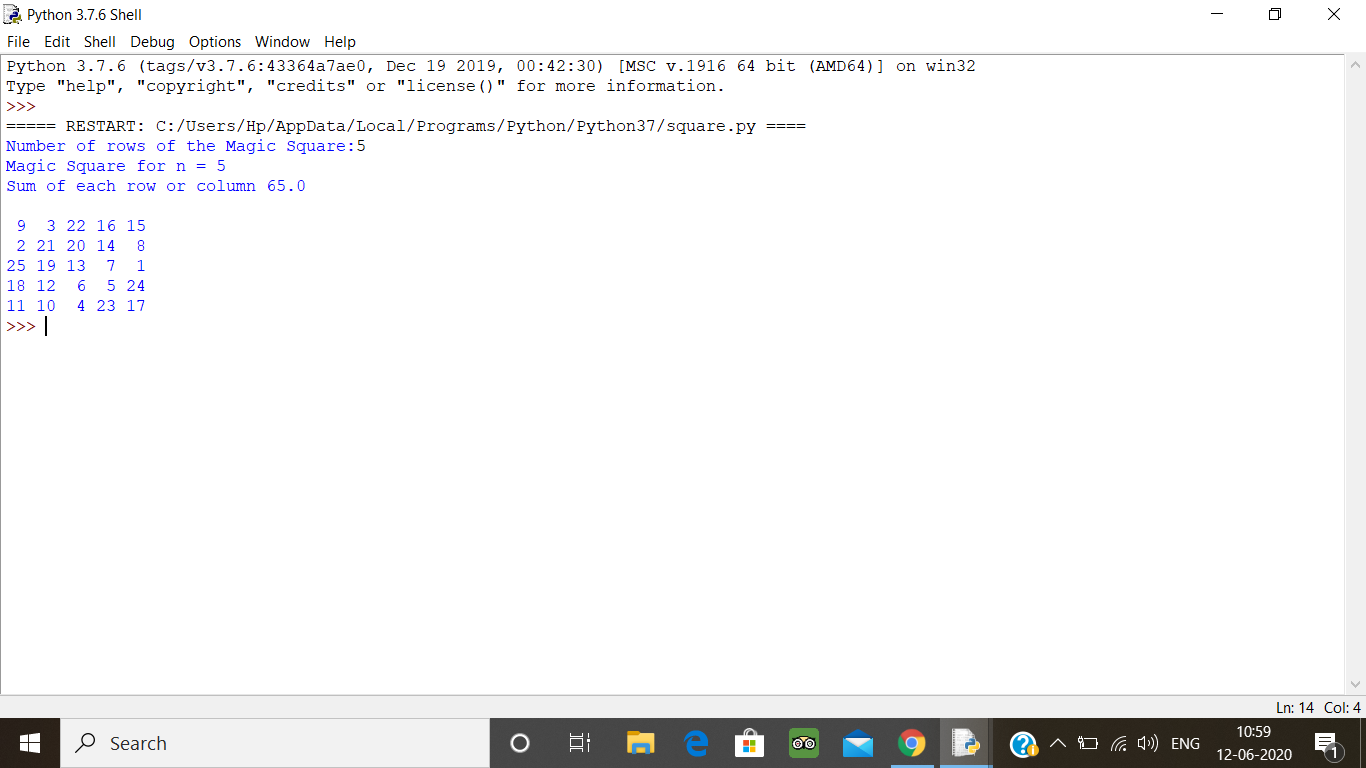
if j == n - 1:

print()

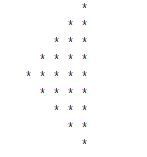
n=int(input("Number of rows of the Magic Square:"))

generateSquare(n)

**output:**



2. Python program to print the pattern



def pattern(n):

    k = 2 \* n - 2

    for i in range(0, n-1):

        for j in range(0, k):

            print(end=" ")

        k = k - 2

        for j in range(0, i + 1):

            print("\* ", end="")

        print("")

    k = -1

    for i in range(n-1,-1,-1):

        for j in range(k,-1,-1):

            print(end=" ")

        k = k + 2

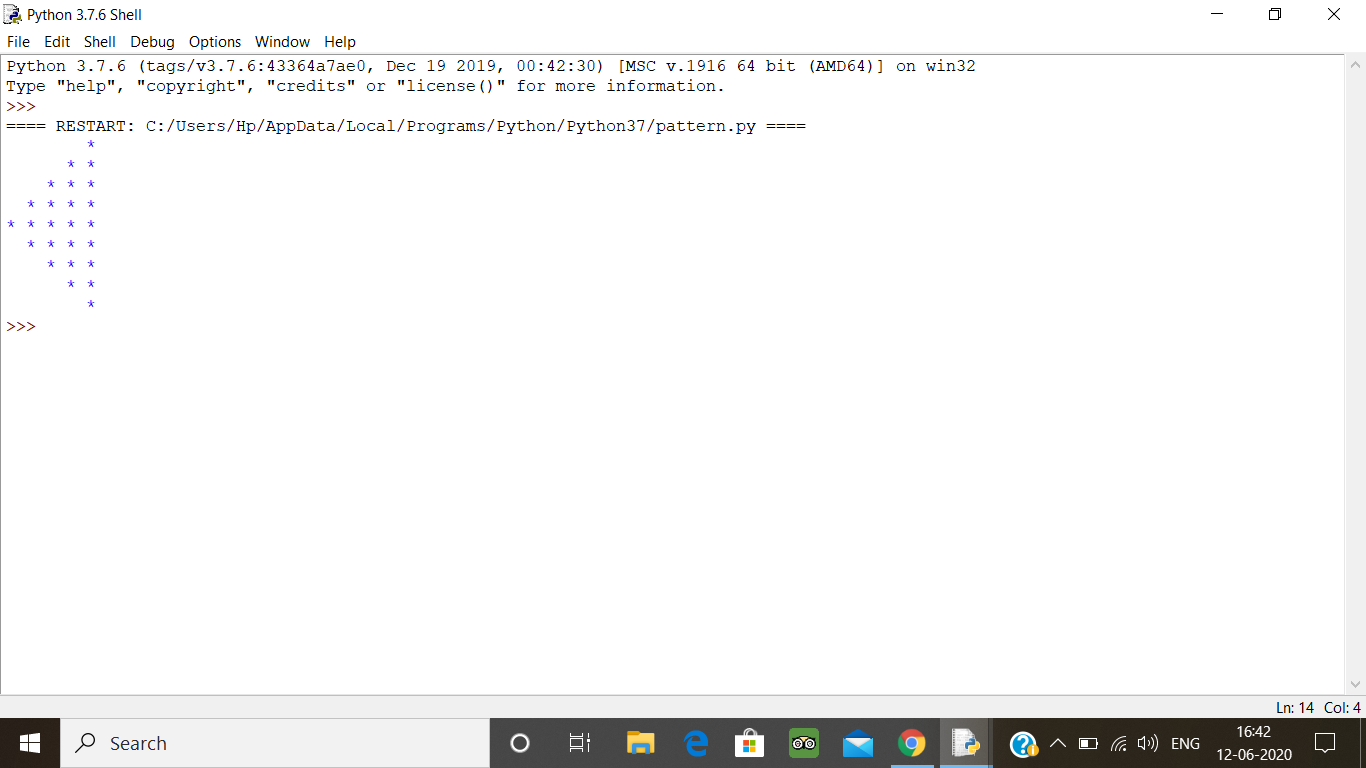
        for j in range(0, i + 1):

            print("\* ", end="")

        print("")

pattern(5)

**output:**



## 3. Write a Java program to find maximum width of 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 an attribute root.  
Root represents the root node of the tree and initializes it to null.  
a. findMaximumWidth() will find out the maximum width of the given binary tree:

Variable maxWidth will store the maximum number of nodes present in any level.  
The queue is used for traversing binary tree level-wise.  
It checks whether the root is null, which means the tree is empty.  
If not, add the root node to queue. Variable nodesInLevel keeps track of the number of nodes in each level.  
If nodesInLevel > 0, remove the node from the front of the queue and add its left and right child to the queue. For the first iteration, node 1 will be removed and its children nodes 2 and 3 will be added to the queue. In the second iteration, node 2 will be removed, its children 4 and 5 will be added to the queue and so on.  
MaxWidth will store max(maxWidth, nodesInLevel). So, at any given point of time, it will represent the maximum number of nodes.  
This will continue till all the levels of the tree is traversed.

import java.util.LinkedList;

import java.util.Queue;

public class Main {

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;

public Main(){

root = null;

}

public int findMaximumWidth() {

int maxWidth = 0;

int nodesInLevel = 0;

Queue<Node> queue = new LinkedList<Node>();

if(root == null) {

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

return 0;

}

else {

queue.add(root);

while(queue.size() != 0) {

nodesInLevel = queue.size();

maxWidth = Math.max(maxWidth, nodesInLevel);

while(nodesInLevel > 0) {

Node current = queue.remove();

if(current.left != null)

queue.add(current.left);

if(current.right != null)

queue.add(current.right);

nodesInLevel--;

}

}

}

return maxWidth;

}

public static void main(String[] args) {

Main bt = new Main();

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.left.left.left = new Node(8);

System.out.println("Maximum width of the binary tree: " + bt.findMaximumWidth());

}

}

**Output:**

