**DAILY ASSESSMENT FORMAT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **18/07/2020** | **Name:** | **Abhishek Vasudev Mahendrakar** | |
| **Course:** | **30 days coding challenge-HackerRank** | **USN:** | **4AL17EC003** | |
| **Topic:** | **Day 11-14** | **Semester & Section:** | **6th-‘A’** | |
| **Github Repository:** | **ECEAbhishekVMahendrakar** | **E-mail:** | **abhi2244mahendrakar@gmail.com** | |
| **FORENOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Report – Report can be typed or hand written for up to two pages.** Day 11: 2D Arrays **Task** Calculate the hourglass sum for every hourglass in A, then print the maximum hourglass sum.  Code:  #!/bin/python3  import math  import os  import random  import re  import sys  def get\_sum(mat,r,c):      s=0      s+=mat[r-1][c-1]      s+=mat[r-1][c]      s+=mat[r-1][c+1]      s+=mat[r][c]      s+=mat[r+1][c-1]      s+=mat[r+1][c]      s+=mat[r+1][c+1]      return s  if \_\_name\_\_ == '\_\_main\_\_':      arr = []      for \_ in range(6):          arr.append(list(map(int, input().rstrip().split())))      \_max\_=-63      for i in range(1,5):          for j in range(1,5):              curr= get\_sum(arr,i,j)              if \_max\_<curr:                  \_max\_= curr      print(\_max\_) Day 12: Inheritance **Task** You are given two classes, *Person* and *Student*, where *Person* is the base class and *Student* is the derived class. Completed code for *Person* and a declaration for *Student* are provided for you in the editor. Observe that *Student* inherits all the properties of *Person*.  Complete the *Student* class by writing the following:   * A *Student* class constructor, which has 4 parameters:   1. A string, firstName.   2. A string, lastName.   3. An integer, id.   4. An integer array (or vector) of test scores, scores. * A *char calculate()* method that calculates a Student object's average and returns the grade character representative of their calculated average:   https://s3.amazonaws.com/hr-challenge-images/17165/1458142706-3073bc9143-Grading.png  **Code:**  class Person:      def \_\_init\_\_(self, firstName, lastName, idNumber):          self.firstName = firstName          self.lastName = lastName          self.idNumber = idNumber      def printPerson(self):          print("Name:", self.lastName + ",", self.firstName)          print("ID:", self.idNumber)  class Student(Person):      #   Class Constructor      #      #   Parameters:      #   firstName - A string denoting the Person's first name.      #   lastName - A string denoting the Person's last name.      #   id - An integer denoting the Person's ID number.      #   scores - An array of integers denoting the Person's test scores.      #      # Write your constructor here        #   Function Name: calculate      #   Return: A character denoting the grade.      #      # Write your function here      def \_\_init\_\_(self, firstName,lastName,idNumber,scores):          Person.\_\_init\_\_(self, firstName, lastName, idNumber)          self.scores = scores      #   Function Name: calculate      #   Return: A character denoting the grade.      def calculate(self):          sum=0          for score in scores:              sum += score          average = sum/len(scores)          if average < 40:              return 'T'          elif average < 55:              return 'D'          elif average < 70:              return 'P'          elif average < 80:              return 'A'          elif average < 90:              return 'E'          else:              return 'O'  line = input().split() Day 13: Abstract Classes **Task** Given a *Book* class and a *Solution* class, write a *MyBook* class that does the following:   * Inherits from *Book* * Has a parameterized constructor taking these 3 parameters:   1. string title   2. string author   3. int price * Implements the *Book* class' abstract *display()* method so it prints these 3 lines:   1. , a space, and then the current instance's title.   2. , a space, and then the current instance's author.   3. , a space, and then the current instance's price.   **Note:** Because these classes are being written in the same file, you must not use an access modifier (e.g.: public) when declaring *MyBook* or your code will not execute.  **Code:**  from abc import ABCMeta, abstractmethod  class Book(object, metaclass=ABCMeta):      def \_\_init\_\_(self,title,author):          self.title=title          self.author=author      @abstractmethod      def display(): pass  #Write MyBook class  class MyBook(Book):      def \_\_init\_\_(self, title, author, price):          Book.\_\_init\_\_(self,title,author)          self.price= price      def display(self):          print("Title:",self.title)          print("Author:",self.author)          print("Price:",str(self.price))  title=input()  author=input()  price=int(input())  new\_novel=MyBook(title,author,price)  new\_novel.display() Day 14: Scope **Task** Complete the *Difference* class by writing the following:   * A class constructor that takes an array of integers as a parameter and saves it to the element instance variable. * A *computeDifference* method that finds the maximum absolute difference between any 2 numbers in N and stores it in the maximumDifference instance variable.   Code:  class Difference:      def \_\_init\_\_(self, a):          self.\_\_elements = a          self.maximumDifference= 0      # Add your code here        def computeDifference(self):            for i in a:              for j in a:                  x= abs(i-j)                  if self.maximumDifference < x:                      self.maximumDifference= x  # End of Difference class  \_ = input()  a = [int(e) for e in input().split(' ')]  d = Difference(a)  d.computeDifference()  print(d.maximumDifference) | | | |