1. Implement a class iterator to flatten a nested list of lists of integers. Each list element is either an integer or a list. There can be many levels of nested lists in lists.

The class initializes with a nested list. It also has two methods:

1. next() returns an integer in the order of appearance.

2. hasNext() returns True / False regarding if all integers have been retrieved or not.

Write the Class implementation for three required methods.

**Examples**

ni, actual = NestedIterator([[1, 1], 2, [1, 1]]), []

while ni.hasNext():

actual.append(ni.next())

actual ➞ [1, 1, 2, 1, 1]

ni, actual = NestedIterator([1, [4, [6]]]), []

while ni.hasNext():

actual.append(ni.next())

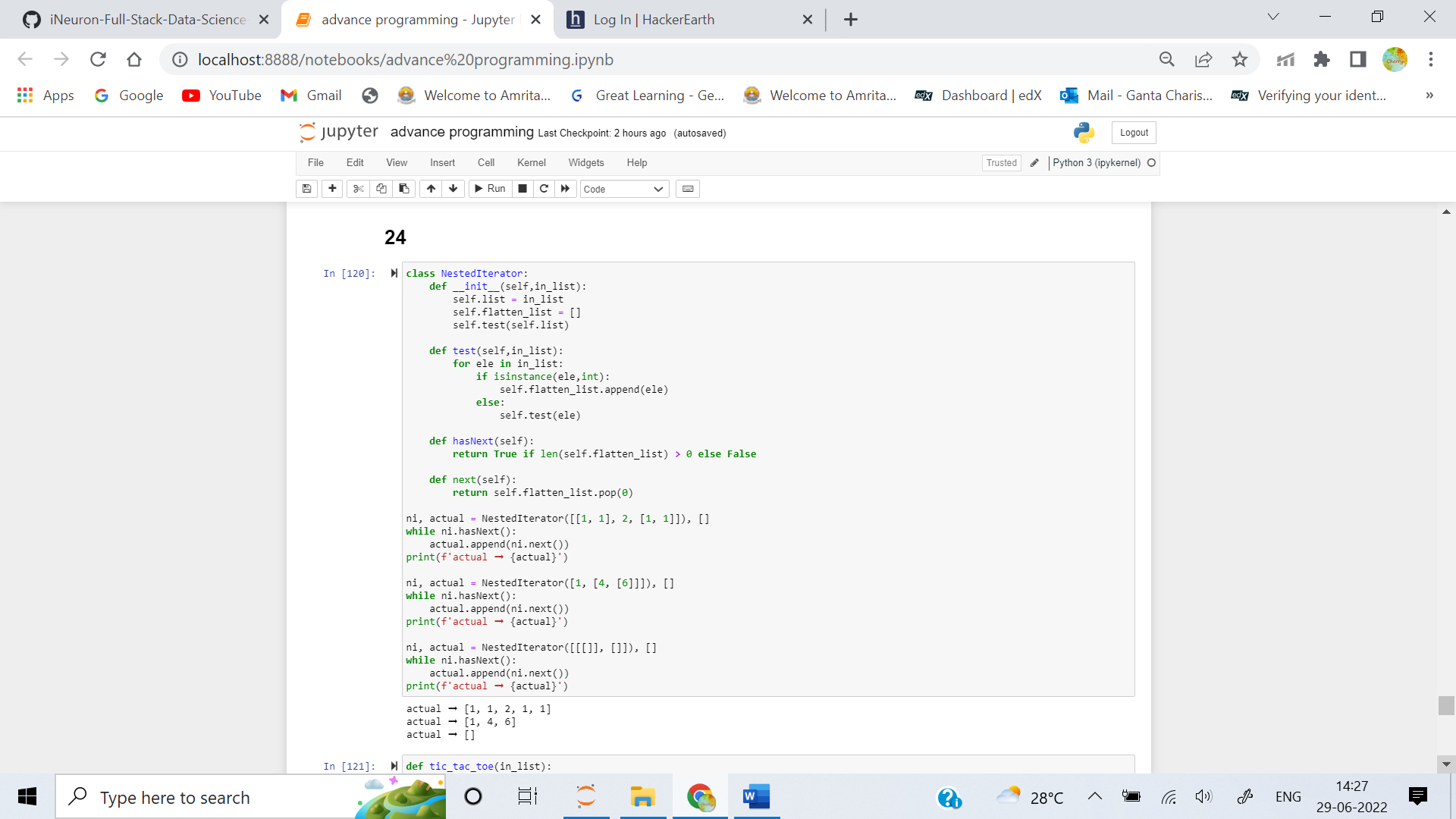
actual ➞ [1, 4, 6]

ni, actual = NestedIterator([[[]], []]), []

while ni.hasNext():

actual.append(ni.next())

actual ➞ []

ANS: 

2. Implement the class Shape that receives perimeter and density function into \_\_init\_\_ method. The list of consecutive corners defines shape of a 2-dimensional object. The density function defines the mass distribution inside the shape. To compute mass in a certain point m(x, y) = small\_square \* density(x, y). The \_\_init\_\_ method calls other internal methods that compute three characteristics of the shape:

- area

- total mass

- center of mass (xc, yc)

The computational grid has distance between two neighboring points as 2 \* delta, the distance between a grid point and the perimeter wall is delta.

**Examples**

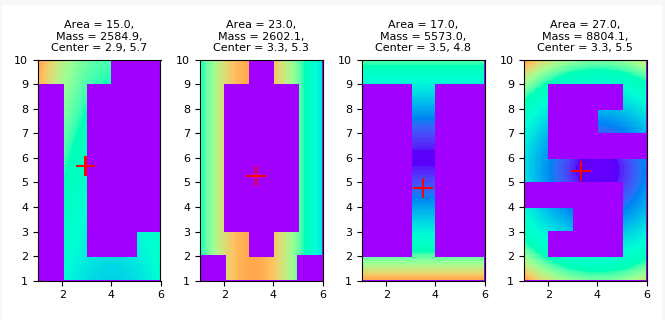
sh\_ex1 = Shape([(1, 1), (3, 1), (3, 2), (1, 2)], lambda x, y: 100 + 100 \* x)

sh\_ex1.area ➞ 2.0

sh\_ex1.mass ➞ 600.0

sh\_ex1.mass\_center ➞ (2.1, 1.5)

The example can be verified via analytical integration. Other shapes in Tests are slightly more complicated and require numerical integration as illustrated here:

ANS: **import** **numpy** **as** **np**

**def** MonteCarlo\_double(f, g, x0, x1, y0, y1, n):

*"""*

*Monte Carlo integration of f over a domain g>=0, embedded*

*in a rectangle [x0,x1]x[y0,y1]. n^2 is the number of*

*random points.*

*"""*

*# Draw n\*\*2 random points in the rectangle*

x = np.random.uniform(x0, x1, n)

y = np.random.uniform(y0, y1, n)

*# Compute sum of f values inside the integration domain*

f\_mean = 0

num\_inside = 0 *# number of x,y points inside domain (g>=0)*

**for** i **in** range(len(x)):

**for** j **in** range(len(y)):

**if** g(x[i], y[j]) >= 0:

num\_inside += 1

f\_mean += f(x[i], y[j])

f\_mean = f\_mean/float(num\_inside)

area = num\_inside/float(n\*\*2)\*(x1 - x0)\*(y1 - y0)

**return** area\*f\_mean

3. Given a 3x3 matrix of a completed tic-tac-toe game, create a function that returns whether the game is a win for "X", "O", or a "Draw", where "X" and "O" represent themselves on the matrix, and "E" represents an empty spot.

Examples

tic\_tac\_toe([

["X", "O", "X"],

["O", "X", "O"],

["O", "X", "X"]

]) ➞ "X"

tic\_tac\_toe([

["O", "O", "O"],

["O", "X", "X"],

["E", "X", "X"]

]) ➞ "O"

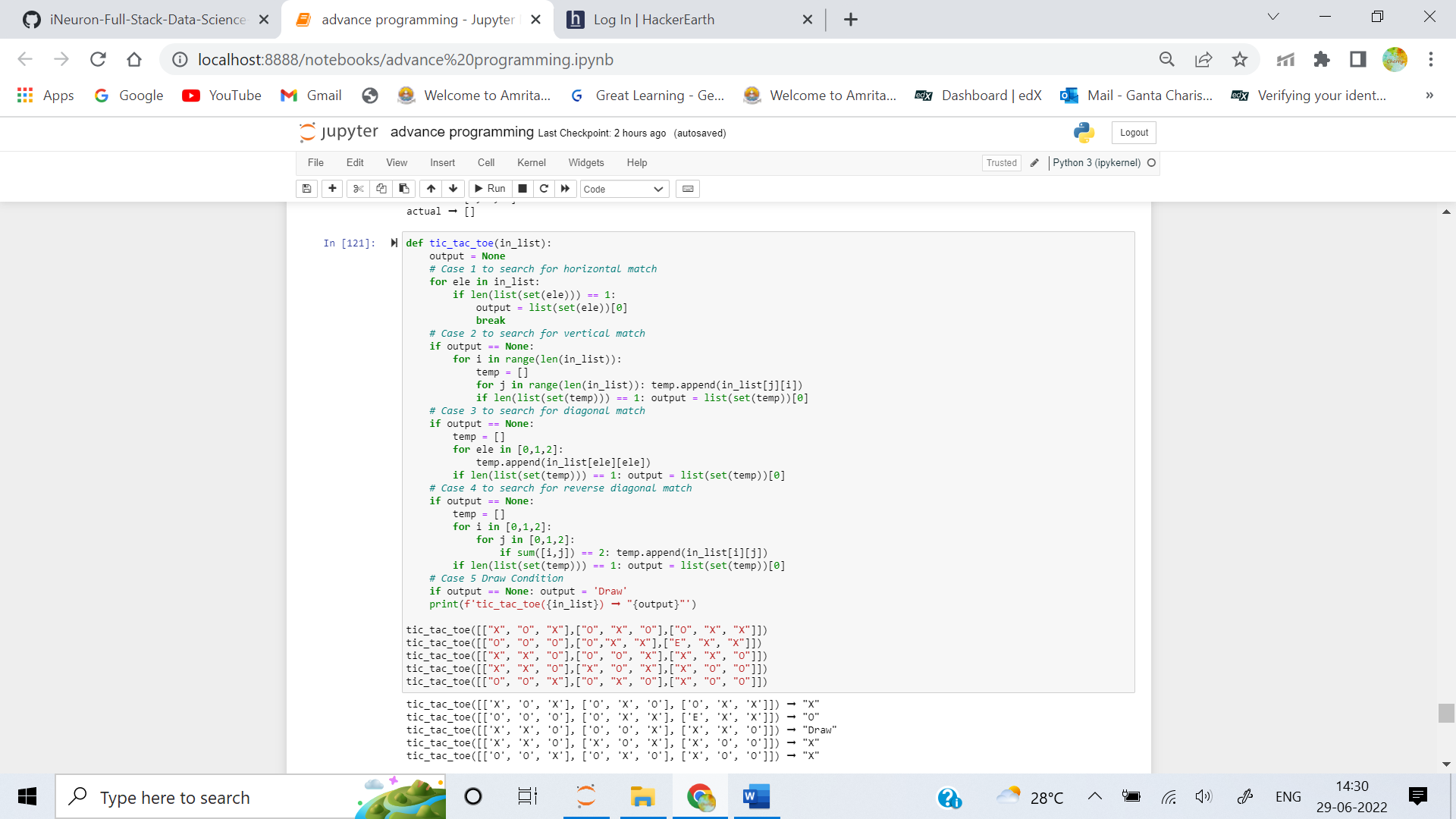
tic\_tac\_toe([

["X", "X", "O"],

["O", "O", "X"],

["X", "X", "O"]

]) ➞ "Draw"

ANS: 

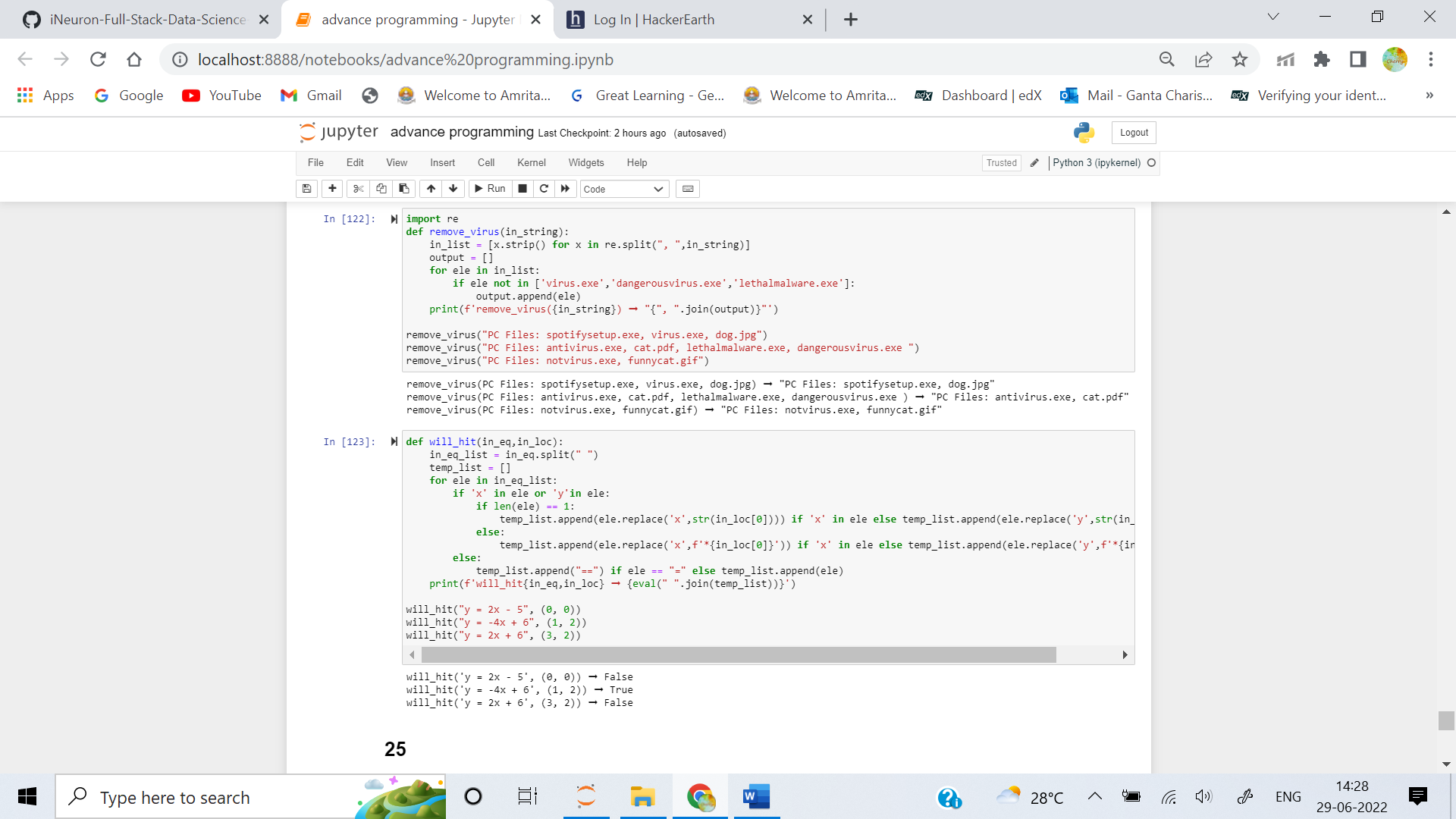
4. Your computer might have been infected by a virus! Create a function that finds the viruses in files and removes them from your computer.

**Examples**

remove\_virus("PC Files: spotifysetup.exe, virus.exe, dog.jpg") ➞ "PC Files: spotifysetup.exe, dog.jpg"

remove\_virus("PC Files: antivirus.exe, cat.pdf, lethalmalware.exe, dangerousvirus.exe ") ➞ "PC Files: antivirus.exe, cat.pdf"

remove\_virus("PC Files: notvirus.exe, funnycat.gif") ➞ "PC Files: notvirus.exe, funnycat.gif")

ANS: 

5. In a video game, a meteor will fall toward the main character's home planet. Given the meteor's trajectory as a string in the form y = mx + b and the character's position as a tuple of (x, y), return True if the meteor will hit the character and False if it will not.

Examples

will\_hit("y = 2x - 5", (0, 0)) ➞ False

will\_hit("y = -4x + 6", (1, 2)) ➞ True

will\_hit("y = 2x + 6", (3, 2)) ➞ False

ANS: 