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     []
```

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

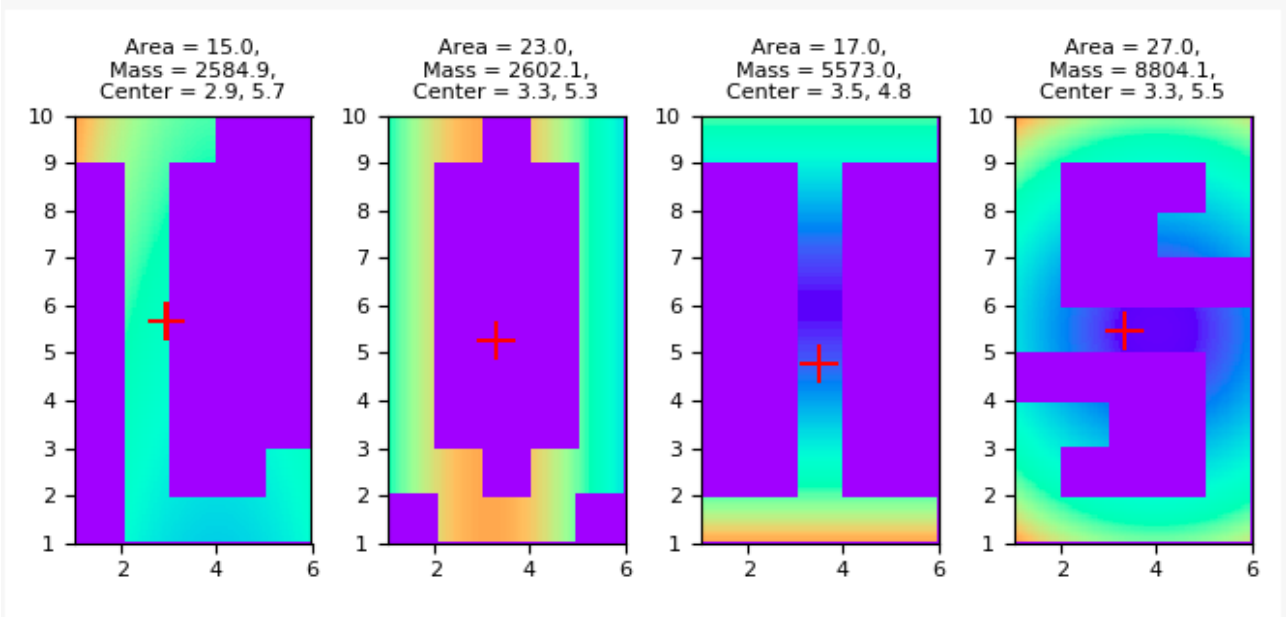
sh_ex1 = Shape([(1, 1), (3, 1), (3, 2), (1, 2)], lambda x, y: 100 + 100 * x)

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:



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"

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")

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