**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 ➞ []

In [1]:

**class** NestedIterator:

**def** \_\_init\_\_(self,in\_list):

self**.**list **=** in\_list

self**.**flatten\_list **=** []

self**.**test(self**.**list)

**def** test(self,in\_list):

**for** ele **in** in\_list:

**if** isinstance(ele,int):

self**.**flatten\_list**.**append(ele)

**else**:

self**.**test(ele)

**def** hasNext(self):

**return** **True** **if** len(self**.**flatten\_list) **>** 0 **else** **False**

**def** next(self):

**return** self**.**flatten\_list**.**pop(0)

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

**while** ni**.**hasNext():

actual**.**append(ni**.**next())

print(f'actual ➞ {actual}')

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

**while** ni**.**hasNext():

actual**.**append(ni**.**next())

print(f'actual ➞ {actual}')

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

**while** ni**.**hasNext():

actual**.**append(ni**.**next())

print(f'actual ➞ {actual}')

actual ➞ [1, 1, 2, 1, 1]

actual ➞ [1, 4, 6]

actual ➞ []

**2. 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"

In [2]:

**def** tic\_tac\_toe(in\_list):

output **=** **None**

*# Case 1 to search for horizontal match*

**for** ele **in** in\_list:

**if** len(list(set(ele))) **==** 1:

output **=** list(set(ele))[0]

**break**

*# Case 2 to search for vertical match*

**if** output **==** **None**:

**for** i **in** range(len(in\_list)):

temp **=** []

**for** j **in** range(len(in\_list)): temp**.**append(in\_list[j][i])

**if** len(list(set(temp))) **==** 1: output **=** list(set(temp))[0]

*# Case 3 to search for diagonal match*

**if** output **==** **None**:

temp **=** []

**for** ele **in** [0,1,2]:

temp**.**append(in\_list[ele][ele])

**if** len(list(set(temp))) **==** 1: output **=** list(set(temp))[0]

*# Case 4 to search for reverse diagonal match*

**if** output **==** **None**:

temp **=** []

**for** i **in** [0,1,2]:

**for** j **in** [0,1,2]:

**if** sum([i,j]) **==** 2: temp**.**append(in\_list[i][j])

**if** len(list(set(temp))) **==** 1: output **=** list(set(temp))[0]

*# Case 5 Draw Condition*

**if** output **==** **None**: output **=** 'Draw'

print(f'tic\_tac\_toe({in\_list}) ➞ "{output}"')

tic\_tac\_toe([["X", "O", "X"],["O", "X", "O"],["O", "X", "X"]])

tic\_tac\_toe([["O", "O", "O"],["O","X", "X"],["E", "X", "X"]])

tic\_tac\_toe([["X", "X", "O"],["O", "O", "X"],["X", "X", "O"]])

tic\_tac\_toe([["X", "X", "O"],["X", "O", "X"],["X", "O", "O"]])

tic\_tac\_toe([["O", "O", "X"],["O", "X", "O"],["X", "O", "O"]])

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"

tic\_tac\_toe([['X', 'X', 'O'], ['X', 'O', 'X'], ['X', 'O', 'O']]) ➞ "X"

tic\_tac\_toe([['O', 'O', 'X'], ['O', 'X', 'O'], ['X', 'O', 'O']]) ➞ "X"

**3. 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")

In [3]:

**import** re

**def** remove\_virus(in\_string):

in\_list **=** [x**.**strip() **for** x **in** re**.**split(", ",in\_string)]

output **=** []

**for** ele **in** in\_list:

**if** ele **not** **in** ['virus.exe','dangerousvirus.exe','lethalmalware.exe']:

output**.**append(ele)

print(f'remove\_virus({in\_string}) ➞ "{", "**.**join(output)}"')

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

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

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

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"

**4. 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

In [4]:

**def** will\_hit(in\_eq,in\_loc):

in\_eq\_list **=** in\_eq**.**split(" ")

temp\_list **=** []

**for** ele **in** in\_eq\_list:

**if** 'x' **in** ele **or** 'y'**in** ele:

**if** len(ele) **==** 1:

temp\_list**.**append(ele**.**replace('x',str(in\_loc[0]))) **if** 'x' **in** ele **else** temp\_list**.**append(ele**.**replace('y',str(in\_loc[1])))

**else**:

temp\_list**.**append(ele**.**replace('x',f'\*{in\_loc[0]}')) **if** 'x' **in** ele **else** temp\_list**.**append(ele**.**replace('y',f'\*{in\_loc[1]}'))

**else**:

temp\_list**.**append("==") **if** ele **==** "=" **else** temp\_list**.**append(ele)

print(f'will\_hit{in\_eq,in\_loc} ➞ {eval(" "**.**join(temp\_list))}')

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

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

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

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

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

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