Iterating through a ragged list is a bit trickier than iterating through a grid with lists of equal length. We would need to determine the length of the current list in the loop before running the nested loop and for each iteration of the nested loop, the length needs to be updated for the current row.
The following is an example of printing every element in the above-ragged list:

```
rows = len(ragged_list)
for row in range(rows):
    cols = len(ragged_list[row]) # now the number of cols depends on each row's length
    print("Row", row, "has", cols, "columns: ", end="")
    for col in range(cols):
        print(ragged_list[row][col], " ", end="")
    print()
```

The following is the output for the code example above:

```
Row 0 has 3 columns: 1  2  3
Row 1 has 2 columns: 4  5
Row 2 has 1 columns: 6
Row 3 has 4 columns: 7  8  9  10
```

Compulsory Task 1

Create a file named **minesweeper.py**

Create a function that takes a grid of # and -, where each hash (#) represents a mine and each dash (-) represents a mine-free spot.

Return a grid, where each dash is replaced by a digit, indicating the number of mines immediately adjacent to the spot i.e. (horizontally, vertically, and diagonally).

Example of an input:

```
[ ["-", "-", "-", "#", "#"],
  ["-", "#", "-", "-", "-"],
  ["-", "-", "#", "-", "-"],
  ["-", "#", "#", "-", "-"],
  ["-", "-", "-", "-", "-"] ]
```

Example of the expected output:

```
[ ["1", "1", "2", "#", "#"],
  ["1", "#", "3", "3", "2"],
  ["2", "4", "#", "2", "0"],
  ["1", "#", "#", "2", "0"],
  ["1", "2", "2", "1", "0"] ]
```

Here is a tip. When checking adjacent positions to a specific position in the grid, the following table might assist you in determining adjacent indexes:

NW position = current_row - 1 current_col - 1	N position = current_row - 1 current_col	NE position = current_row - 1 current_col + 1
W position = current_row current_col - 1	Current position = current_row current_col	E position = current_row current_col + 1
SW position = Current_row + 1 current_col - 1	S position = current_row +1 current_col	SE position = current_row + 1 current_col + 1

Also ensure that when checking adjacent positions in the grid that you take into account that on the edges of the grid, you may go out of bounds.

Lastly, you could make use of the enumerate function in Python to keep track of the index points and values without having to create a count variable and explicitly iterate the count variable to keep track of the current row or column index.

Below is an example of how the enumerate function works.

```
#list to be iterated through
values = ["a", "b", "c"]

# "count" here is used to keep track of the index point
# "value" is the value of the current element in the loop
# The enumerate method takes 2 arguments the iterable and the starting
# value for “count” which we set at 0 to represent the index of the first
# index in the list.
for count, value in enumerate(values, start = 0):
    print(f'Index {count} contains the value {value}')

Below is the output generated:
Index 0 contains the value a
Index 1 contains the value b
Index 2 contains the value c
```

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we’ve done a good job?

Click here to share your thoughts anonymously.