Two Dimensional Lists

Introduction

- A Two-dimensional list is a list of lists.
- It represents a table/matrix with rows and columns of data.
- In this type of list, the position of a data element is referred to by two indices instead of one.

Consider an example of recording temperatures 4 times a day, 4 days in a row. Such data can be presented as a two-dimensional list as below.

```
Day 1 - 11, 12, 5, 2
Day 2 - 15, 6, 10, 6
Day 3 - 10, 8, 12, 5
Day 4 - 12, 15, 8, 6
```

The above data can be represented as a two-dimensional list as below.

```
T = [[11, 12, 5, 2], [15, 6, 10, 6], [10, 8, 12, 5], [12, 15, 8, 6]]
```

Accessing Values in a Two Dimensional list

The data elements in two-dimensional lists can be accessed using two indices. One index referring to the main or parent list (inner list) and another index referring to the position of the data element in the inner list. If we mention only one index then the entire inner list is printed for that index position.

The example below illustrates how it works.

```
T = [[11, 12, 5, 2], [15, 6, 10, 6], [10, 8, 12, 5], [12,15,8,6]]
print(T[0]) #List at index 0 in T
print(T[1][2]) #Element at index 2 in list at index 1 in T
```

When the above code is executed, it produces the following result –

```
[11, 12, 5, 2]
10
```

Updating Values in Two Dimensional list

We can update the entire inner list or some specific data elements of the inner list by reassigning the values using the list index.

```
T = [[11, 12, 5, 2], [15, 6, 10, 6], [10, 8, 12, 5], [12, 15, 8, 6]]
```

This can be done the same way as in 1D lists. Consider the examples given below:

```
T[0][1]= 1#Update the element at index 1 of list at index 0 of T to 1
T[1][1]= 7
```

This would modify the list as:

```
T= [[11, 1, 5, 2], [15, 7, 10, 6], [10, 8, 12, 5], [12, 15, 8, 6]]
```

Inserting Values in a Two Dimensional list

We can insert new data elements at any specific position by using the `insert()` method and specifying the index.

In the below example a new data element is inserted at index position 2.

```
T = [[11,12,5,2],[15,6,10,6],[10,8,12,5],[12,15,8,6]]
T.insert(2, [0,5,11,13])
```

This would modify the list as:

```
T = [[11,12,5,2],[15,6,10,6],[0,5,11,13],[10,8,12,5],[12,15,8,6]]
```

List Comprehension

List comprehensions are used for creating new lists from other iterable sequences. As list comprehensions return lists, they consist of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

This is the basic syntax:

```
new_list = [expression for_loop_one_or_more conditions]
```

Finding squares of numbers in a list with the help of list comprehensions:

```
numbers = [1, 2, 3, 4]
squares = [n**2 for n in numbers]
print(squares)  # Output: [1, 4, 9, 16]
```

Here, square brackets signify that the output is a list. `n**2` is the expression executed for each element and `for n in numbers` is used to iterate over each element. In other words, we execute `n**2` (expression) for each element in `numbers`.

Find common numbers from two lists using list comprehension:

```
list_a = [1, 2, 3, 4]
list_b = [2, 3, 4, 5]
common_num = [a for a in list_a for b in list_b if a == b]
print(common_num) # Output: [2, 3, 4]
```

Here, we iterate over the two lists `list_a` and `list_b` and if a in `list_a` is equal to b in `list_b`, only then we add it to out output list.

Jagged Lists

Till now we have seen lists with an equal number of columns in all rows. Jagged lists are the two-dimensional lists with a variable number of columns in various rows. Various operations can be performed on such lists, the same way as in 2D lists with the same number of columns throughout. Some examples of jagged lists are shown below:

```
T1 = [[11,12,5,2],[10,6],[10,8,12],[12]]
T2 = [[1,2,3], 1, 3, [1,2]]
T3 = [[1],[],[1,2,1],0]
```

The elements in a jagged list can be accessed the same way as shown:

```
>>> T1[3]
[12]
>>> T1[0][2]
5
```

Note: Keep in mind the range of columns for various rows. Indexes out of the range can produce `indexOutOfRange` error.

Printing a Two Dimensional List

There are many ways to **print a 2D List**. Some of them are discussed below:

Simply Print the List

We can simple print a given list in a single line. This can be done by just using the `print()` function. This is shown as below:

```
T= [[11,12,5,2],[15,6,10,6],[0,5,11,13],[10,8,12,5],[12,15,8,6]]
print(T)
```

We get the output as:

```
[[11,12,5,2],[15,6,10,6],[0,5,11,13],[10,8,12,5],[12,15,8,6]]
```

Printing in Multiple Lines Like a Matrix

We can use nested `for` loops to print a 2D List. The outer loop iterates over the inner lists and the inner nested loop prints the elements of the lists. This can be shown as follows:

```
T= [[11,12,5,2],[15,6,10,6],[0,5,11,13],[10,8,12,5],[12,15,8,6]]
for row in T: #Every row in T is a list
    for col in row: #Every col is an element of the list row
        print(col, end=" ") #Print col
    print() #New Line
```

The above code will print T as:

```
11 12 5 2
15 6 10 6
0 5 11 13
10 8 12 5
12 15 8 6
```

Using List Comprehension and `.join()` method

`.join()`: The `.join()` string method returns a string by joining all the elements of an iterable sequence, separated by a string separator. The working is shown below:

```
>>> "-".join("abcd")
a-b-c-d
>>> "-".join(['a','b','c','d'])
a-b-c-d
```

The `join()` method produced a new string `"a-b-c-d"` from the given string and list.

Now, we can use this string method to print 2D Lists.

```
T= [[11,12,5,2],[15,6,10,6],[0,5,11,13],[10,8,12,5],[12,15,8,6]]
for row in T: #Every row in T is a list
    output = " ".join([str(col) for col in row])
    print(output)
```

The output will be:

```
11 12 5 2
15 6 10 6
0 5 11 13
10 8 12 5
12 15 8 6
```

Input Of Two Dimensional Lists

We will discuss two common ways of taking user input:

1. **Line Separated Input-** Different rows in different lines.
2. **Space Separated Input-** Taking input in a single line.

Line Separated Input:

We will use list comprehension to take user input. For every row, we take a list as input. The syntax would be:

```
n = int(input())
input= [[int(col) for col in input.split()] for row in range n]
print(input)
```

User Input:

```
5
11 12 5 2
15 6 10 6
0 5 11 13
10 8 12 5
12 15 8 6
```

Output:

```
[[11,12,5,2],[15,6,10,6],[0,5,11,13],[10,8,12,5],[12,15,8,6]]
```

Space Separated Input:

For this we will require the user to specify the number of rows (say `row`) and columns (say `col`) of the 2D list. The user will then enter all the elements of the 2D list in a single line. We will first convert this input stream to a list (say `inputL`). The next step would be using list comprehension to make a 2D list (say `fnlList`) from this list. It can be observed that `fnlList[i,j]=inputL[i*row+j]`. The python code for this can be written as follows:

```
row = int(input()) #Number of rows
col = int(input()) #Number of columns
inputL= input().split() #Converting input string to list
fnlList=[[int(inputL[i*col+j]) for j in range(col)] for i in range(row)]
print(fnlList)
```

User Input:

```
4
5
11 12 5 2 15 6 10 6 0 5 11 13 10 8 12 5 12 15 8 6
```

Output:

```
[[11,12,5,2],[15,6,10,6],[0,5,11,13],[10,8,12,5],[12,15,8,6]]
```