PythonCourse_8_2DLists

April 18, 2021

0.0.1 Two Dimentional Lists

[6]:
```
## Creating 2D Lists

# clm-   #0 #1 #2 #3
# row- 0# 1  2  3  4
# row- 1# 5  6  7  8
# row- 2# 9  10 11 12
# row- 3# 13 14 15 16

li = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]

#Accessing 2DList elements

print(li[2][2]) #Important : [a][b] not [a,b]
print(li[0][0])
#print(li[0,0]) #TypeError: list indices must be integers or slices, not tuple
#print(li[3][6]) #IndexError: list index out of range
```

```
11
1
```

[8]:
```
## Storing 2D Lists

#2D List is a list of lists
#2D Lists store the references of the lists
print('li',id(li))
print('li0',id(li[0]))
print('li1',id(li[1]))
print('li2',id(li[2]))
print('li3',id(li[3]))
print('li01',id(li[0][1]))
```

```
li 86475392
li0 83641728
li1 81524160
li2 86194432
li3 80851712
li01 8791414875968
```

[9]:
```
## 2D Lists are Mutable

#Since lists store the references, they are mutable
li[0][1] = 5
print(li)
```

```
[[1, 5, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
```

[10]:
```
## Jagged Lists

# Jagged lists are 2D lists whose column sizes are not same
li_jag = [[1,2,3],[4,5],[6,7,8,9,10]]
print(li_jag[0])
print(li_jag[1])
print(li_jag[2])
```

```
[1, 2, 3]
[4, 5]
[6, 7, 8, 9, 10]
```

List Comprehension #New Lists can be created in a single line using List Comprehension new_list = [output (condition) expression condition]

[17]:
```
#new list having square of each element of list

li = [1,2,3,4,5,6]

# Using append method
new_list_append = []
for element in li:
    new_list_append.append(element**2)
print(new_list_append)

# Using list comprehension
new_list_compre = [element**2 for element in li]
print(new_list_compre)
```

```
[1, 4, 9, 16, 25, 36]
[1, 4, 9, 16, 25, 36]
```

[18]:
```
#list comprehension with condition
#new list having square of even elements of list

new_list = [element**2 for element in li if element%2==0]
print(new_list)
```

```
[4, 16, 36]
```

[19]:
```
#list comprehension with more conditions
#new list having elements which are multiple of 2 and 3

new_list = [element for element in li if element%2==0 if element%3==0]
print(new_list)
```

```
[6]
```

[20]:
```
#list comprehension using many for loops
#new list having elements common in given two lists

li1 = [1,2,3,4,5,6,7,8]
li2 = [3,6,4,2,0,11,16]
new_list = [element1 for element1 in li1 for element2 in li2 if element1 ==␣
 ↪element2]
print(new_list)
```

```
[2, 3, 4, 6]
```

[21]:
```
#list comprehension having if else condition
#new list having element square for multiples of 2 and only element for others

new_list = [element**2 if element%2==0 else element for element in li]
print(new_list)
```

```
[1, 4, 3, 16, 5, 36]
```

[98]:
```
#new list having characters of the string

string = 'Jayashree'
new_list_str = [element for element in string]
print(new_list_str)
```

```
['J', 'a', 'y', 'a', 's', 'h', 'r', 'e', 'e']
```

[24]:
```
#list comprehension to generate list of lists
#new list having the list elements into separate lists within a list

li_str = ['Jayashree','Rekha','Srinivasan']
new_list_str = [[element] for element in li_str]
print(new_list_str)
```

```
[['Jayashree'], ['Rekha'], ['Srinivasan']]
```

[25]:
```
#new list having the list elements into separate lists within a list

li_str = ['Jayashree','Rekha','Srinivasan']
new_list_str = [[char for char in element] for element in li_str]
print(new_list_str)
```

```
[['J', 'a', 'y', 'a', 's', 'h', 'r', 'e', 'e'], ['R', 'e', 'k', 'h', 'a'], ['S',
'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n']]
```

Other Concepts in 2D Lists

[99]:
```
## Input of 2D Lists - Format 1

#Get n(rows) and m(columns)
#Get the elements in each row (row by row)
#Can be used for regular 2D lists as well as jagged lists

string = input().split()
n,m = int(string[0]),int(string[1])

list_2d = [[int(element) for element in input().split()] for i in range(n)]
print(list_2d)
```

```
3 4
1 2 3 4
5 6 7 8
9 10 11 12
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

[103]:
```
## Input of 2D Lists - Format 2

#Get n(rows) and m(columns)
#Get elements in a single line
#element (i,j) --> element (m * i) + j in the line

string = input().split()
n,m = int(string[0]),int(string[1])

line_2d = input().split()
print('2D Line',line_2d)
list_2d = [ [int(line_2d[m * i + j]) for j in range(m)] for i in range(n)]
print('2D List',list_2d)
```

```
3 4
1 2 3 4 5 6 7 8 9 10 11 12
2D Line ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
2D List [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

[104]:
```
## Input of 2D Lists - Format 3

#Get n(rows) and m(columns) and elements in single line
#element (i,j) --> element (m * i) + j in the line

string = input().split()
n,m = int(string[0]),int(string[1])
```

```
line_2d = string[2:] #slice the string list
print('2D Line',line_2d)
list_2d = [ [int(line_2d[m * i + j]) for j in range(m)] for i in range(n)]
print('2D List',list_2d)
```

```
3 4 1 2 3 4 5 6 7 8 9 10 11 12
2D Line ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
2D List [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

[47]:
```
## Iterating on a 2D List - Row wise

#Print the elements of a 2D List- using row and column no
#Can be used to print Regular 2D lists
li = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
n = 3
m = 4 #Cannot be used for jagged lists as m is variable in jagged
for i in range(n):
    for j in range(m):
        print(li[i][j], end = ' ')
    print()
```

```
1 2 3 4
5 6 7 8
9 10 11 12
```

[48]:
```
#Print the elements of a 2D List- using list element
#Can be used to print Regular 2D lists and Jagged Lists
li = [[1,2,3,4],[5,6],[7,8,9]]

for row in li:
    for element in row:
        print(element, end = ' ')
    print()
```

```
1 2 3 4
5 6
7 8 9
```

[57]:
```
## Join keyword

#Used for printing
#Can be used on those which are iteratable ie, lists and strings
print('ab'.join('abc')) # Joins 'ab' after every char in string but not to the␣
 ↪last
#print('ab'.join([1,2,3])) # TypeError: sequence item 0: expected str instance,␣
 ↪int found
                           # Can use join only for string type
```

```
print('ab'.join(['1','2','3']))
print(' '.join(['1','2','3']))
print(type(' '.join(['1','2','3']))) #Join returns a string
```

```
aabbabc
1ab2ab3
1 2 3
<class 'str'>
```

[106]:
```
#Print the elements of a 2D List- using join keyword
del str
li = [[1,2,3,4],[5,6],[7,8,9]]
for row in li:
    print(' '.join([str(ele) for ele in row]))
```

```
1 2 3 4
5 6
7 8 9
```

[90]:
```
## Iterating on a 2D List - Columnwise

## Print the column index with highest sum - using n
def ColSumHighest(li):
    highest_col_sum = -1
    highest_col_sum_index = -1

    n = len(li) # no of elements in li is row no
    m = len(li[0]) #no of elements in any row is the column no
    for j in range(m):
        col_sum = 0
        for i in range(n):
            col_sum += li[i][j]
        #print(col_sum)
        if col_sum > highest_col_sum:
                highest_col_sum = col_sum
                highest_col_sum_index = j
    return highest_col_sum_index, highest_col_sum

li = [[1,2,3,4],[5,6,7,8],[9,10,11,7]]
highest_col_sum_index, highest_col_sum = ColSumHighest(li)
print('Highest Column Sum Index', highest_col_sum_index)
print('Highest Column Sum', highest_col_sum)
```

```
Highest Column Sum Index 2
Highest Column Sum 21
```

[91]:
```
## Iterating on a 2D List - Columnwise

## Print the column index with highest sum - using list elements
```

```
def ColSumHighest(li):
    highest_col_sum = -1
    highest_col_sum_index = -1

    n = len(li) # no of elements in li is row no
    m = len(li[0]) #no of elements in any row is the column no
    for j in range(m):
        col_sum = 0
        for row in li: #taking the element column wise in the list element
            col_sum += row[j]
        #print(col_sum)
        if col_sum > highest_col_sum:
                highest_col_sum = col_sum
                highest_col_sum_index = j
    return highest_col_sum_index, highest_col_sum

li = [[1,2,3,4],[5,6,7,8],[9,10,11,7]]
highest_col_sum_index, highest_col_sum = ColSumHighest(li)
print('Highest Column Sum Index', highest_col_sum_index)
print('Highest Column Sum', highest_col_sum)
```

```
Highest Column Sum Index 2
Highest Column Sum 21
```