Composite Types, Lists Part 2

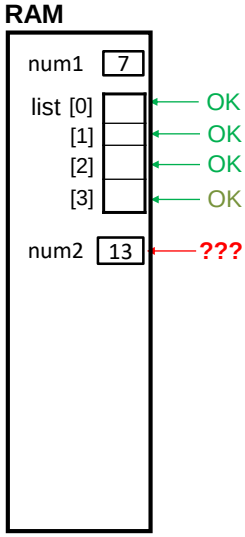
- Style: avoiding list bound exceptions (overflow)
- When to use multi-dimensional lists
- Creating 2D lists statically and dynamically
- How to access a 2D list and its parts
- Basic 2D list operations: display, accessing parts, copying the list

Take Care Not To Exceed The Bounds Of The List

Example: `7listBounds.py`

```
num1 = 7
list = [0, 1, 2, 3]
num2 = 13
for i in range (0, 4, 1):
    print (list [i])


print()
print(list [4])  ←—???
```



A Common Way To Avoid Overflowing A List

- Use a **constant** in conjunction with the list.

```
SIZE = 100
```

- The value in the constant controls traversals of the list

```
for i in range (0, SIZE, 1):
     myList[i] = int(input ("Enter a value:" ))

for i in range (0, SIZE, 1):
    print(myList [i])
```

A Common Way To Avoid Overflowing A List (2)

- Use a constant in conjunction with the list.

```
SIZE = 100000
```

- The value in the constant controls traversals of the list

```
for i in range (0, SIZE, 1):
     myList [i] = int(input ("Enter a value:" ))

for i in range (0, SIZE, 1):
    print (myList [i])
```

A Python Specific Approach To Avoid Overflow

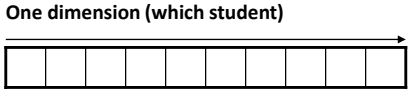
- Use the length function `len` to get the length of list.
  - Since a function call requires some resources/time it's a bit more efficient to store the length in a variable.
  - Unless the length of the list changes refer to the variable rather than calling function again.
- Example:

```
myList = someFunctionCreatesList()
myListLength = len(myList)
i = 0
while (i < myListLength):
    print(myList[i])
```

James Tam

When To Use Lists Of Different Dimensions

- It's determined by the data – the number of categories of information determines the number of dimensions to use.
- Examples:
- (1D list)
  - Tracking grades for a class (previous example)
  - Each cell contains the grade for a student i.e., `grades[i]`
  - There is one dimension that specifies which student's grades are being accessed



- (2D list)
  - Expanded grades program
  - Again there is *one dimension* that specifies which student's grades are being accessed
  - The *other dimension* can be used to specify the lecture section

When To Use Lists Of Different Dimensions (2)

- (2D list continued)

Student

Lecture section

	First student	Second student	Third student	…
L01				
L02				
L03				
L04				
L05				
:				
L0N				

When To Use Lists Of Different Dimensions (3)

- (2D list continued)
- Notice that each row is merely a 1D list
- (A 2D list is a list containing rows of 1D lists)

Important:

- List elements are specified in the order of `[row] [column]`
- Specifying only a single set of brackets specifies the row

Columns (e.g. grades)

	[0]	[1]	[2]	[3]
[0] L01				
[1] L02				
[2] L03				
[3] L04				
[4] L05				
[5] L06				
[6] L07				

Rows (e.g. lecture section)

Creating And Initializing A Multi-Dimensional List In Python (Fixed Size During Creation)

General structure

```
<list_name> = [ [<value 1>,  <value 2>, ... <value n>],
                [<value 1>,  <value 2>, ... <value n>],
                     : :        :
                     : :        :
                [<value 1>,  <value 2>, ... <value n>] ]
```

Rows

Columns

Creating And Initializing A Multi-Dimensional List In Python (2): Fixed Size During Creation

Name of the example program: `8display2DList.py`

Learning: creating, displaying a fixed size 2D list

```
matrix = [ [0, 0, 0],
           [1, 1, 1],
           [2, 2, 2],
           [3, 3, 3]]
```

```
      c=0  c=1  c=2
r = 0 [0,  0,  0]
r = 1 [1,  1,  1]
r = 2 [2,  2,  2]
r = 3 [3,  3,  3]
```

```
for r in range (0, 4, 1):
   print (matrix[r])  #Each call to print displays a 1D list
```

```
for r in range (0,4,1):
   for c in range (0,3,1):
       print(matrix[r][c], end="")
   print()
```

```
  0 1 2 (col)
r = 0 000
r = 1 111
r = 2 222
r = 3 333
```

#List element

```
print(matrix[2][0])   #2 not 0
```

```
2
```

2D Lists: Levels Of Access

```
matrix = [ [0, 0, 0],
           [1, 1, 1],
           [2, 2, 2],
           [3, 3, 3]]
print (matrix)  #Entire list
print(matrix[0]) #First row
print(matrix[3][1]) #4th row, 2nd column
print(matrix[0][0][0]) #What does this do?
```

```
[[0, 0, 0], [1, 1, 1], [2, 2, 2], [3, 3, 3]]
[0, 0, 0]
3
TypeError: 'int' object is not subscriptable
```

```
matrix = [ [["a","b"], 0, 0],
           [1, 1, 1],
           [2, 2, 2],
           [3, 3, 3]]

print(matrix[0][0][0]) #Now what does this do?
```

James Tam

In Python: List Elements Need Not Store The Same Data Type

- This is one of the differences between Python lists and arrays in other languages
- Example:

```
aList = [False, "James", "Tam", "210-9455", 707, 10.5]
```

Copying Lists

- Important: A variable that appears to be a list is really a reference to a list.
  - Recall: the reference and the list are two separate memory locations!

```
matrix = [ [0, 0, 0],
           [1, 1, 1],
           [2, 2, 2],
           [3, 3, 3]]
```

  - Wrong way to 'copy' a 2D list

`aList1 = aList2` (Why is this wrong? Hint: recall what is stored in `aList1` and `aList1`)

James Tam

Copying Lists: Example

- **Name of the example program:** `9copyingListsBothWays.py`
- This is the **wrong way**.

```
aGrid1 = create()
aGrid2 = aGrid1
aGrid1[3][3] = "!"
print("First list")
display(aGrid1)
print("Second list")
display(aGrid2)
```

```
# FYI:
def create():
    aGrid = [["*","*","*","*"],
             ["*","*","*","*"],
             ["*","*","*","*"],
             ["*","*","*","*"]]
    return(aGrid)
```

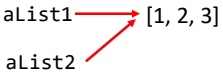
James Tam

New Terminology

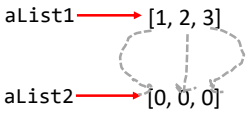
- **Shallow copy**: copies what's stored in the reference (location of a list).

Code

```
aList1 = [1,2,3]
aList2 =aList1
```



- **Deep copy**: copies the data from one list to another.
  - Create a new list e.g. aList2 = [0]*3
  - Copy each piece of data (list elements) from one list to another e.g. `aList2[0] = aList1[0]`



James Tam

Copying Lists: Example (2)

- This is the right way.

```
aGrid1 = create()
aGrid2 = create()
copy(aGrid1,aGrid2)
```

```
def copy(destination,source):
    for r in range (0,SIZE,1):
        for c in range (0,SIZE,1):
            destination[r][c] = source[r][c]
```

```
copy(aGrid1,aGrid2)
aGrid1[0][0] = "?" #These statements prove there's two lists
aGrid1[3][3] = "?"
print("First list")
display(aGrid1)
print("Second list")
display(aGrid2)
```

James Tam

Copying Lists: Write The Code Yourself

- For this class you should not use the list `copy` method.
- Not all programming languages have this capability (you will need to know how to do it yourself).
- Writing the code yourself will provide you with extra practice.

James Tam

Extra Practice

List operations:

- For a numerical list: implement some common mathematical functions (e.g., average, min, max, mode – last one is challenging).
- For any type of list: implement common list operations (e.g., displaying all elements one at a time, inserting elements at the end of the list, insert elements in order, searching for elements, removing an element, finding the smallest and largest element).
  - In order to develop your programming skills you should write the code yourself rather than using predefined python methods such as `append`, `min`, `max` etc.

After This Section You Should Now Know

- When to use lists of different dimensions
- Basic operations on a 2D list
- Techniques to avoid overflowing the bounds of a list
- How to create a 2D list: fixed size and by dynamically creating it
- How to access a 2D list: the whole list, rows in the list and individual elements
- Python lists need not be homogenous (contain the same type of element)
- How to properly copy the contents of a 2D list into another 2D list as well as a common mistake when copying lists