## Composites: Lists

- Some list methods
- When to use multi-dimensional lists
- Creating 2D lists
- How to access a 2D list and its parts
- Basic 2D list operations: display, accessing parts, copying the list
- Using named constants to stay within list bounds
- Dynamically creating 2D lists with `append`.

James Tam

## What You Should Already Know

- The following notes were already covered in the looping/repetition section.
- They are included for your reference (and if needed to remind you of what you need to review).
- We won't be covering them again in class but instead we will immediately proceed to the next section.

James Tam

Cover after midterm

# New Type Of Variable: List

- This is only a very basic introduction.
  - For the keeners: more details will come later.
- **String**: consists of individual elements that can be accessed via an index (zero to length of the string minus one) `s1 = "Jim tam"`

```
0 1 2 3 4 5 6
Jim tam
```

- **List:** need not consist only of characters nor does it have to be homogeneous (e.g. all integers, all Booleans)
  - i.e. Python lists can be heterogeneous
  - `list1 = [1, "a",True]`

```
0   1   2
1   a   True
```

James Tam

Cover after midterm

# Creating A List (Fixed Size)

**•Format ('n' element list):**

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

Element 0 Element 1 Element n-1

**Example:**

```
#List with 5 elements, index ranges from 0 to (5-1)

percentages = [50.0, 100.0, 78.5, 99.9, 65.1]
               0     1      2     3     4
```

**Other Examples:**

```
letters = ["A", "B", "A"]
names = ["The Borg", "Klingon ", "Hirogin", "Jem'hadar"]
```

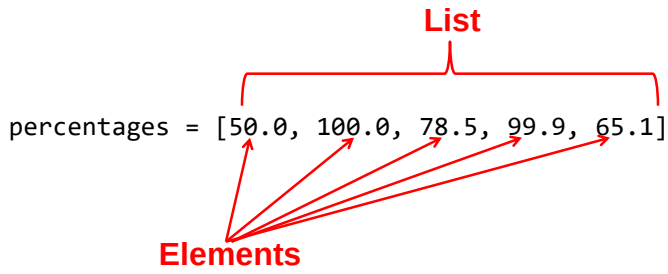
1 These 4 names (Borg, Klingon, Hirogin, Jem'hadar) © are CBS

James Tam

Cover after midterm

## Accessing/Displaying A List

- Because a list is composite you can access the entire list or individual elements.




- Name of the list accesses the whole list
  `print(percentages)`

```
>>> print(percentages)
[50.0, 100.0, 78.5, 99.9, 65.1]
```

- Name of the list and an index "`[index]`"accesses an element
  `print(percentages[1])`

```
>>> print(percentages[1])
100.0
```



Cover after midterm

## Basic List Operations

- **Name of the online example**:
  `5modifying_displaying_list`
- Common list operations:
  - **Create a new fixed size list:**

```
aList = [2,6,2]
```

  - **Displaying entire list:**

```
i = 0
size = len(aList)
while(i < size): #i takes on values from 0 – (size-1)
    print(aList[i], end=" ")
    i = i + 1
```

  - **Modifying a single element**

```
aList[size-1] = 3
```

  - **Modifying all elements**

```
while(i < size): #i takes on values from 0 – (size-1)
    aList[i] = aList[i] * 2
    i = i + 1
```

Cover after midterm

## Additional List Operations

- **Name of the online example**: `6adding_2_end_modify_select_while`

```
aList = ["A","a","z","B"]
```

New list operations:

- **Adding new elements: adding new elements to end** (append method):

```
aList.append(ch)
```

- **Modifying select elements (based upon a condition):**

```
i = 0
size = len(aList)
while(i < size): #A=ASCII 65, Z=90
    if((aList[i]>="A") and (aList[i]<="Z")):
        aList[i] = aList[i] + "!" #Applies to caps only
    i = i + 1
```

James Tam

Cover after midterm

## For Loops Can Be Used To Iterate Lists

- **Name of the online example**: `7adding_2_select_for`

```
aList = ["A","a","z","B"]
```

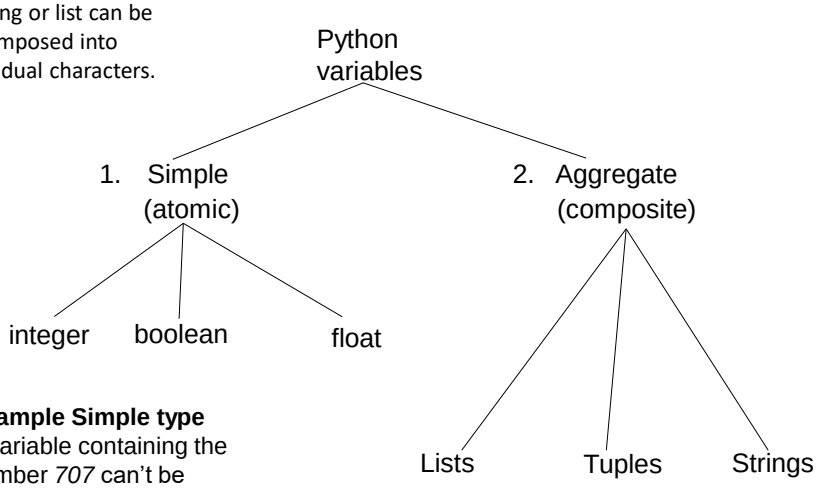
- Iterating list using a for-loop:

```
for ch in aList:
    print(ch)
```

James Tam

## Types Of Variables

**How lists fit into what you have learned**
A string or list can be decomposed into individual characters.




**Example Simple type**
A variable containing the number *707* can't be meaningfully decomposed into parts

James Tam

## Some Topics (Sort Of) Covered

James Tam

## Basic List Operations

- Common list operations:
  - **Create a new fixed size list:**

```
aList = [2,6,2]
```

  - **Displaying entire list:**

```
i = 0
size = len(aList)
while(i < size): #i takes on values from 0 – (size-1)
    print(aList[i], end=" ")
    i = i + 1
```

  - **Modifying a single element**

```
aList[size-1] = 3
```

  - **Modifying all elements**

```
while(i < size): #i takes on values from 0 – (size-1)
    aList[i] = aList[i] * 2
    i = i + 1
```

  - **Adding new elements: adding new elements to end** (append method):

```
aList.append(ch)
```

James Tam

## Negative Indices

- Although Python allows for negative indices (-1 last element, -2 second last…-<size>) this is unusual and this approach is not allowed in other languages.
- So unless otherwise told your **index should be a positive integer** ranging from <zero> to <list size – 1>
- **Don't use negative indices**.

James Tam

## Creating A Variable Sized List: Looping

You already know: create a fixed size all at once.

- You can use a loop and **append new elements** onto the end of the list.
- **Name of the full example:** `1list_creation_loop_N_append.py`

```
SIZE = 5
aList = [] #Create new list
i = 0
while(i<SIZE):
    aList.append(i)
    print(aList[i],end=" ")
    i = i + 1
```

```
aList = [] #Create new list
for i in range(0,SIZE,1):
    aList.append(i)
    print(aList[i],end=" ")
```

```
0 1 2 3 4
0 1 2 3 4
```

James Tam

## Creating A Variable Sized List: Repetition Operator

- Create a list of any size but it is initialized all with the same value.
- **Name of the full example:** `2list_creation_repetition.py`

```
SIZE = 5
aList = [0]*SIZE
print(aList)
```

```
[0, 0, 0, 0, 0]
```

```
aList[0] = 7
aList[SIZE-1]= 13
print(aList)
```

```
[7, 0, 0, 0, 13]
```

James Tam

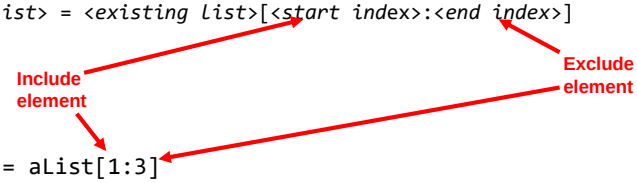
## Some List Methods

- Methods we cover in this version of CPSC 217/231:
  - Example starting list: `aList = [1,2,3]`
  - (Already covered), add to the end: **append**
  - **Insert:** add element at the specified index:
    `insert(<index>,<element>)`
    `aList.insert(0,"first")` `['first', 1, 2, 3]`
  - **Extend:** add another list to the end of another list.
    `extend(<list>))`
    `aList.append(["second","third"])` `['first', 1, 2, 3, ['second', 'third']]`
- Here's some online documentation on these and other list methods:
  - https://www.w3schools.com/python/python_lists_methods.asp
  - If you click 'next' at the above link it also provides basic multiple choice questions to evaluate your knowledge (keep in mind some questions are really simple).

James Tam

## The "Slicing-Operator" (Also Works With Strings)

- The "square brackets" with a single integer accesses/modifies a single element.
- A range can be specified to retrieve multiple elements into a new list "sub list"
- **Format:**
  `<new sub list> = <existing list>[<start index>:<end index>]`




- **Example:**
  `subList1 = aList[1:3]`

James Tam

## Examples Of List Slicing

- **Name of the online example:** `3listSlicing`

```
                0   1   2   3   4   5
aList = ["a","b","c","d","e","f"]
subList1 = aList[1:3]
print(subList1)   ['b', 'c']

subList2 = aList[:3]
print(subList2)
                  ['a', 'b', 'c']
subList3 = aList[4:]
print(subList3)   ['e', 'f']

subList4 = aList[:]
print(subList4)   ['a', 'b', 'c', 'd', 'e', 'f']
```

```
['b', 'c']
['a', 'b', 'c']
['e', 'f']
['a', 'b', 'c', 'd', 'e', 'f']
```

James Tam

## Common List Operations

- They have already been covered previously e.g. creating new empty list, iterating a list, changing/accessing elements etc.
- You can refer to your notes when lists were first introduced (looping/repetition).

James Tam

## Creating A New List By Copying An Existing List

- This is not a comprehensive list
- Assume we have this list:

```
list1 = [1,2,3]
```

  - **Method 1 (python specific):** Utilize one of the prebuilt python methods for copying a list (if you don't know which one to use then use "deep copy).
  - **Method 2 (python specific):** write the code yourself using a FOR-loop

```
for element in list1:
    list2.append(element)
```

**(Hudson and Zhao)**

```
for i in range(0, len(names)):
```

JT: 'names' is a list

  - **Method 3(language independent):** write the code yourself using a WHILE-loop.

```
i = 0
List2 = []
size = len(list1)
while(i<size):
    list2.append(list1[i])
    i = i + 1
```

James Tam

## Creating A New List Via Copying (Python Specific)

- **Name of the full example:** `4list_creation_copying.py`

```
original = [1,2,3]
copy = [i for i in original]
print(original,copy)
```

[1, 2, 3] [1, 2, 3]

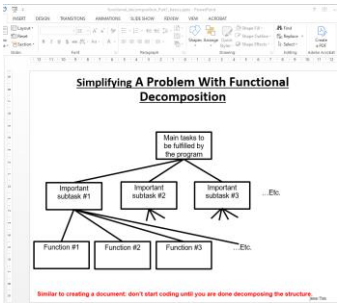
James Tam

## Deciphering The Previous Example

```
original = [1,2,3]
copy = [i for i in original]
print(original,copy)
```

Tam's reaction when first seeing this: WTH?

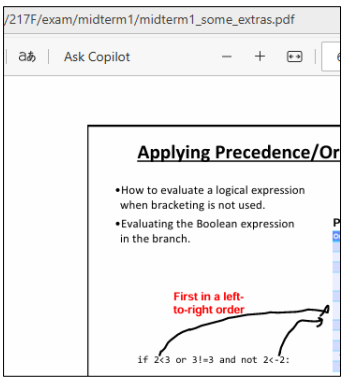
- Remember what I taught you at the beginning of the term, when faced with a complex and/or big problem break it into parts.
- Example: decomposing your program into parts, each part will be implemented with one or more functions.




James Tam

## Reminder: Past Examples Of Decomposition

- Midterm I help materials: tackle a complex Boolean expression by examining only a part at a time.

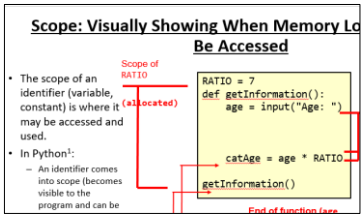



James Tam

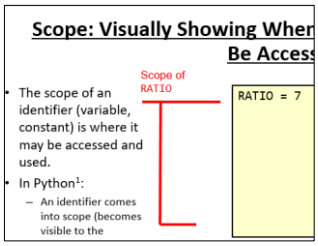
# Reminder: Past Examples Of Decomposition (2)

- Functional decomposition: scope

**Original: Huh???!!!**




**Simplified: You already know this**




James Tam

# Back To The WTH Code

- `copy = [i for i in original]`

`for i in original` — 1) You should know what's happening: from the repetition section: for-loops iterating a list

`i for i in original` — 2) Now that you have been reminded as to what's happening in the code above #1 we explain in class what this is doing

`copy = [i for i in original]` — 3) Finally the result of the expression from #2 above is assigned to 'copy'

James Tam

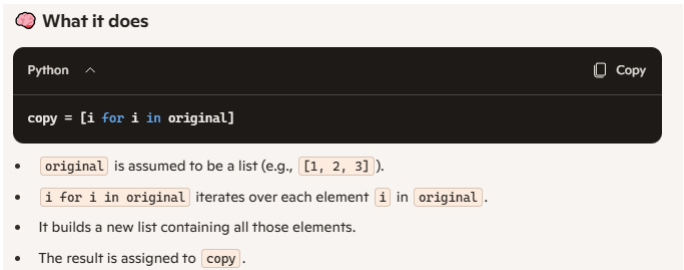
## But What Do I Do When Tam Isn't Around In The Future?

- Rejoice and dance in the streets! (Hallelujah!)
- (Just kidding folks!)

James Tam

## Searching The Web Directly May Be Challenging

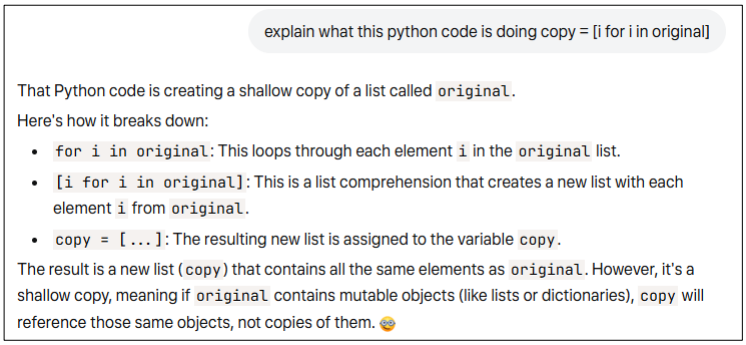
- It may be okay if you have a search term but in this case you just have a concept (non-CPSC courses) or code (in this example).
- Here's where online A.I. tools 'may' help:
  - Bing Co-pilot: "`explain what this python code is doing copy = [i for i in original]`"




James Tam

## Answer From An Alternative 'Intelligence'

- Facebook's Meta:




- JT's caution: Just like web searches confirm your information:
  - Verify by looking at a good source: textbook published by a reputable publisher.
  - Writing your own code and testing the results.

James Tam

## Review: Passing Parameters Which Aren't Lists

- A copy of the **data** stored in the variable is passed into the function.
- For previous examples refer to functional decomposition examples where simple types were passed as function parameters: `bool, int, float`
- **Changes** made to the parameters are **only made to local variables**.

```
def fun(num):
    num = 21  #Only num local to fun changed
num = 12
fun(num)  #Still 12
```

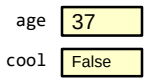
- The changed local variables must have their values returned back to the caller in order to be retained.

James Tam

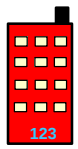
## More Details On Lists

- With the simple variable types (integer, float, boolean) you can think of as a single memory location.
  - E.g.

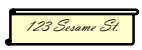
```
age = 37
cool = False
```




- Declaring a list variable will result in two memory locations allocated in memory.
  - One location is for the list itself ("The multi-suite building")




  - Another location "refers to" or contains the address of the building.




James Tam

## Example: Illustrating List References

- **Name of the example program:** `5listReferences.py`

```
num = 123
list1 = [1,2,3]
list2 = list1
print(list1)
print(list2)
```

```
Before
[1, 2, 3]
[1, 2, 3]
```

```
list1[0] = 888
list2[2] = 777
print(list1)
print(list2)
```

```
After
[888, 2, 777]
[888, 2, 777]
```

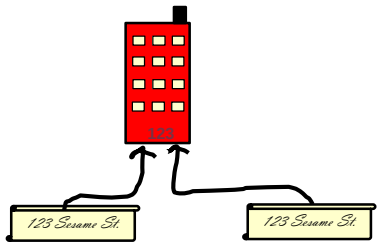
James Tam

## New Term: Shallow Copy

- **Shallow copy:** the address of a list (or some of the composite types) is copied rather than the data (elements of a list).
  - Result: two references containing the address of (references to) a single list.
  - Previous example:

Copy contents (address not a list) from '`list1`' into '`list2`'

```
list2 = list1
```




- The shallow copy allows access to the list so it may be changed by either reference to it (in this example it's '`list1`' or '`list2`').

James Tam

## One Part Of The Previous Example Was Actually Unneeded

```
def read(classGrades):

        :               :

    return(classGrades, average)
```

When list is 'passed' as a parameter…

…returning the list is likely not needed

**More details on 'why' coming up shortly!**

James Tam

## Passing References (Lists)

- Recall: A list variable is actually just a reference to a list (~a paper with an address written on it).

```
aList          = [1,2,3]
```

Reference to the list (contains the memory address)

The list (no name just a location in memory)

- A copy of the address is passed into the function (~copying what's on the paper, the address).

```
def fun(copyList):
    copyList[0] = 10
```

- The local reference 'refers' to the original list (use the paper to go to the specified address).
  - In essence: parameter passing with lists is a shallow copy.

James Tam

## Passing A List As A Parameter

- **A reference to the list is passed**, in the function **a local variable** which is another reference can allow **access to the list**.

Example:

```
def read(classGrades):
    ...
    for i in range (0, CLASS_SIZE, 1):
        temp = i + 1
        print("Enter grade for student no.", temp, ":")
        classGrades[i] = float(input (">"))
        total = total + classGrades[i]

def start():
    classGrades = initialize()
    read(classGrades)
```

James Tam

## Example: Passing Lists As Parameters

- **Name of the example program**: `6listParametersPassAddress`
- Learning : a list parameter allows changes to the original list (persist even after the function ends).

```
def fun1(aListCopy):
    aListCopy[0] = aListCopy[0] * 2
    aListCopy[1] = aListCopy[1] * 2
    return(aListCopy)

def fun2(aListCopy):
    aListCopy[0] = aListCopy[0] * 2
    aListCopy[1] = aListCopy[1] * 2
```

James Tam

## Example: Passing Lists As Parameters (2)

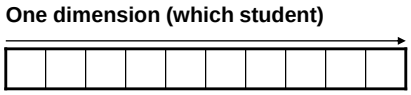
```
def start():
    aList = [2,4]
    print("Original list in start() before function
           calls:\t", end="")
    print(aList)
    aList = fun1(aList)
    print("Original list in start() after calling fun1():\t",
           end="")
    print(aList)
    fun2(aList)
    print("Original list in start() after calling fun2():\t",
           end="")
    print(aList)

start()
```

```
Original list in start() before function calls: [2, 4]
Original list in start() after calling fun1():  [4, 8]
Original list in start() after calling fun2():  [8, 16]
```

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 (table: grades for multiple lectures)
  - 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

James Tam

## 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 | | | | |

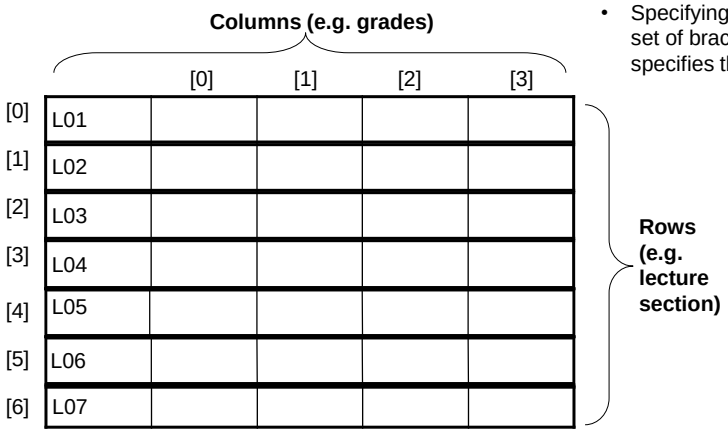
James Tam

# 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




James Tam

# 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

James Tam

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

**Name of the example program**: `7display2DList.py`

Learning: creating, displaying a fixed size 2D list

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

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

for r in range (0,4,1):
    for c in range (0,3,1):#Displays list element
        print(table[r][c], end="")
    print()

print(table[2][0]) #Displays 2 not 0
```

```
    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]
```

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

2
```

James Tam

## Creating 2D Lists Via The Repetition Operator

**Name of the example program**: `8creatingListViaRepetition.py`

Learning:

- Creating a variable sized 2D list using the repetition operator and the append method.
- The 2D list is created by creating a 1D list and **appending the 1D list to the end of the 2D list**.

```
MAX_COLUMNS = 5
MAX_ROWS = 3
ELEMENT = "*"
aList = []
r = 0
while(r < MAX_ROWS):
   tempList = [ELEMENT] * MAX_COLUMNS
   aList.append(tempList)
   r = r + 1
```

James Tam

## Using The Repetition Operator On 1D Lists

- You have just seen how the repetition operator can create a 1D list (one set of square brackets).
- Example:

```
list2 = [0] * 4
print(list2)   [0, 0, 0, 0]
```

- Using a repetition operator on an existing 1D list merely repeats instances of that 1D list.
  - Consider: `list2 = [0] * 4` **`#Repeats list with 1 element x 4`**
  - Examples:
    - `list1= [1,2,3]`
    - `list2 = [0] * 4`
    - `list3 = list1 * 5` **`#List with 15 elements: 3 elements x 5`**
    - `list4 = list2 * 2` **`#List with 8 elements: 4 elements x 2`**

    - `print(len(list3),len(list4))` `15 8`
    - `print(list3)` `[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]`
    - `print(list4)` `[0, 0, 0, 0, 0, 0, 0, 0]`

James Tam

## How To Avoid Overflowing 2D Lists

For 231

- Employ named constants
- Recall that the previous example declared 2 named constants.

```
MAX_COLUMNS = 5
MAX_ROWS = 3
```

- Control access to list elements using these constants.

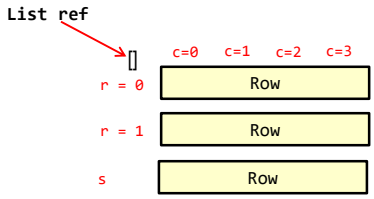
```
r = 0
while(r < MAX_ROWS):
    c = 0
    while(c < MAX_COLUMNS):
        print(aList[r][c], end = "")
        c = c + 1
    print()
    r = r + 1
```

James Tam

# Creating And Initializing A Multi-Dimensional List In Python: Dynamic Creation

**General structure (Using loops)**:

- Create a variable that refers to an empty list
- Create list:
  - One loop (outer loop) traverses the rows.
  - **Each iteration of the outer loop creates a new 1D list (empty at start)**
  - **Then the inner loop traverses the elements of the newly created 1D list** creating and **initializing each element** in a fashion similar to how a single 1D list was created and initialized (add to end)
- Repeat the process for each row in the list




Etc.

```
aGrid = []
for r in range (0, 3, 1):
    aGrid.append ([])
    for c in range (0, 3, 1):
        aValue = <Some source>
        aGrid[r].append(aValue)
```

James Tam

# Repeating Just The Steps In The Code Creating The List

1. Create a variable that refers to an empty list
   ```
   aGrid = []
   ```
2. Successively create rows in the list
   ```
   for r in range (0,noRows,1):
       aGrid.append ([])
   ```
3. Each row is a 1D list, add elements to the end of the 1D list (empty list needed in #2 so that the append method can be called to **add elements to the end**).
   ```
   for c in range (0,noColumns,1):
       aGrid[r].append("*")
   ```

- The [r] part of specifies which row the loop will add elements on the end.
  ```
  aGrid[r].append("*")
  ```

Recall 'append' is unique to a list. Append won't work if for something other than a list but for a list an empty row can have new elements appended.

```
num = 123
num.append(4)
```

James Tam

## Example 2D List Program: A Variable Sized 2D List (Dynamic)

- **Name of the example program:** `5variableSize2DList.py`

```
aGrid = []
noRows = int(input("Number rows: "))
noColumns = int(input("Number columns: "))
#Create list
for r in range (0,noRows,1):
    aGrid.append ([]) #Create empty row, add to list
    for c in range (0,noColumns,1):
        element = input("Type in a single character: ")
        aGrid[r].append(element) #Add to the end of new row
#Display list
for r in range (0,noRows,1):
   for c in range (0,noColumns,1):
      print(aGrid[r][c], end="")
   print()
```

James Tam

## 2D Lists: Using Append

Final JT hint: Make sure you apply the right operation on the right type of variable.

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

table.append([2,1,7]) #Where was the append occurring?
print(table)

table[3].append(3) #Where was the append occurring?
print(table)

#What element is the append applied to?
table[2][1].append(888)

Hint: add the following before the last instruction
print(table[2][1])
print(type(table[2][1]))
```

James Tam

## Lists: Final Notes

- Reminder: python list elements need not be all the same type.
- Python 2D lists need not be rectangular.

```
aList = [[1,True,"hi"],
         [1,2.3],
         []]
```

**Row index 0:** int, bool, string
**Row index 1:** int, float
**Row index 2:** empty list

James Tam

## After This Section You Should Now Know

- When to use lists of different dimensions.
- Basic operations on a 2D list.
- How to create a 2D list: fixed size and a variable sized list by using the repetition operator.
- How to access a 2D list: the whole list, rows in the list and individual elements.
- The use of a named constant to ensure that list boundaries are adhered to.
- The ability to dynamically creating 2D lists using the append function for both the rows and columns.

James Tam

## Copyright Notification

- Unless otherwise indicated, all images in this presentation were provided courtesy of James Tam.