ARRAYS

Mahsa Ziraksima
ACIBADEM UNIVERSITY
Fall 2025

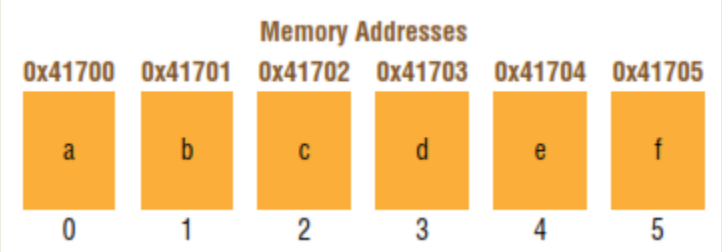
SUMMARY

- Array Performance
- Creating an Array
- Moving Zeros
- Combining Two Lists
- Finding the Duplicates in a List
- Finding the Intersection of Two Lists

ARRAY

- It is a data structure that stores elements with indexes in a contiguous block of memory.
- This block consists of items stored one after the other in your computer's memory.

homogeneous and static



- A Python list is a heterogeneous variable-length (dynamic) array.
- Different programming languages use different indexing schemes.
  - *Python and C use zero- based indexing*
  - *MATLAB and Fortran, use one-based indexing (the first element is index 1)*

ARRAY

- The memory location of the first element in an array is called its base address.
- When adding a new item to an array or finding an item in an array.

```
base_address + index * size_of_data_type
```

- Regardless of whether an array is one-dimensional or multidimensional, your computer stores its elements in a contiguous block of memory.
  - *Access and modify any single element in an array in constant time.*
  - *Searching an unsorted array is O(n)*
  - *Searching an sorted array is O(log n)*
  - *Modifying the shape of an array, like adding or deleting items is O(n)*

ARRAY

```
array = [1, 2, 3]
print(array[0])

array[1] = 'hello'
print(array)
```

```
1
[1, 'hello', 3]
```

```
multi_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(multi_array[1][2])
```

```
6
```

```
import array

arr = array.array('f', (1.0, 1.5, 2.0, 2.5))
print(arr[1])
```

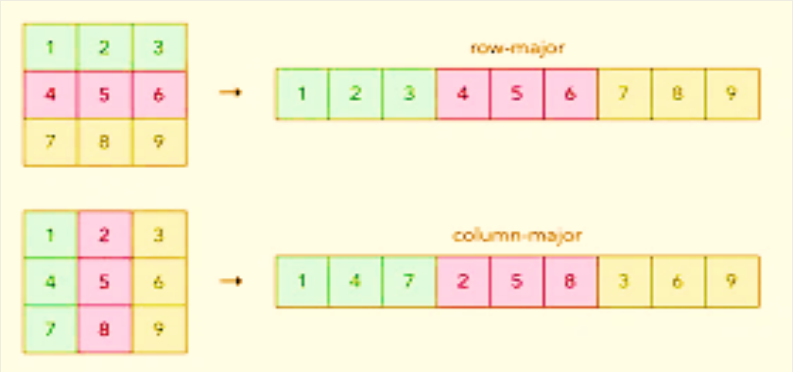
```
1.5
```

```
arr[1] = 'hello'
```

```
TypeError: must be real number, not str
```

ARRAY

- N-dimensional arrays
  - $[[L_1 \ldots U_1][\ldots][L_n \ldots U_n]]$
  - Number of elements = $((U_1 - L_1 + 1) \times \cdots \times (U_n - L_n + 1))$
  - Memory usage = $((U_1 - L_1 + 1) \times \cdots \times (U_n - L_n + 1)) \times m$
- Two-dimensional arrays are also sometimes called matrices
  - $[[L_1 \ldots U_1]\,[L_2 \ldots U_2]]$
  - Number of elements = $((U_1 - L_1 + 1) \times (U_2 - L_2 + 1))$
- In memory:



ARRAY

- A triangular matrix is a special type of square matrix where all of the elements above or below the main diagonal are zero.

Upper Triangular Matrix

$$\begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix}$$

$A[i, j] = 0 \qquad i < j$

Lower Triangular Matrix

$$\begin{pmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{pmatrix}$$

$A[i, j] = 0 \qquad i > j$

$$\begin{bmatrix} 1 & 6 & 7 \\ 0 & 2 & 5 \\ 0 & 0 & 4 \end{bmatrix} \Longrightarrow$$

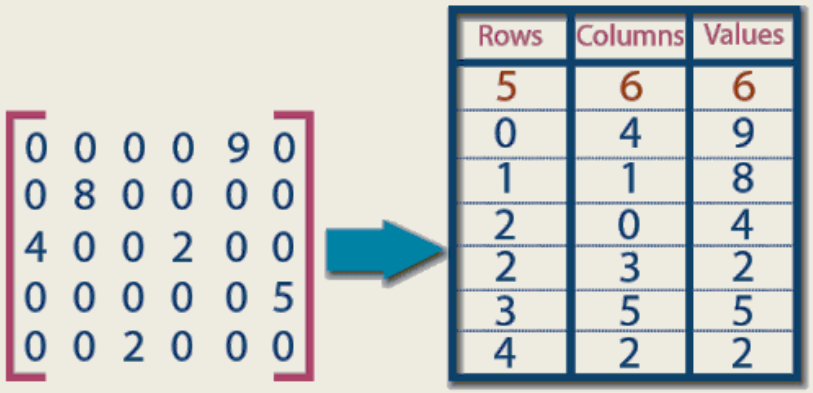
1	2	3	4	5	6
1	6	2	7	5	4

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 5 & 0 \\ 3 & 1 & 1 \end{bmatrix} \Longrightarrow$$

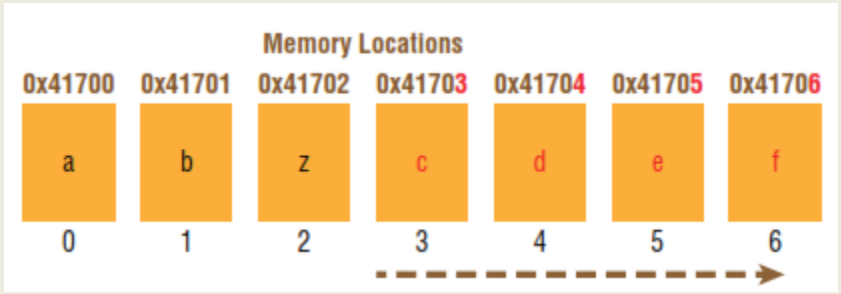
1	2	3	4	5	6
1	2	5	3	1	1

ARRAY

- Sparse matrix:



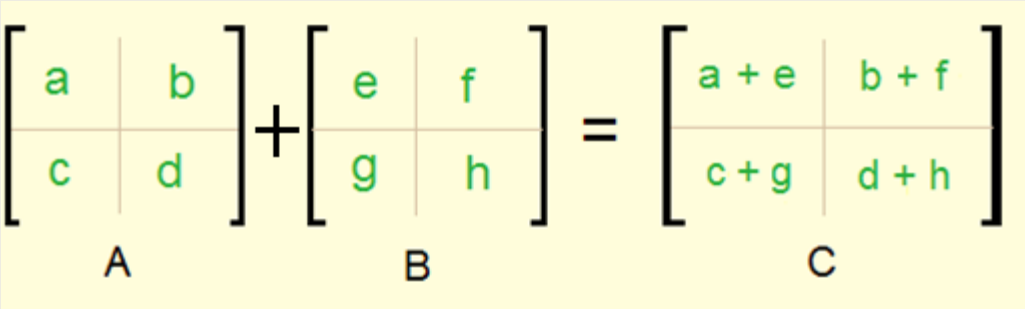
ARRAY



- This problem is worse for static arrays because your computer can't guarantee the memory addresses that come after the memory block it reserved for the array are free.
  - *C reserves a new block of memory for the array, copy all the items from the old block to the new one, add the new element, and then free the old block.*
  - *Python uses overallocation: reserving more memory for an array than it needs.*

ARRAY

- Summation:



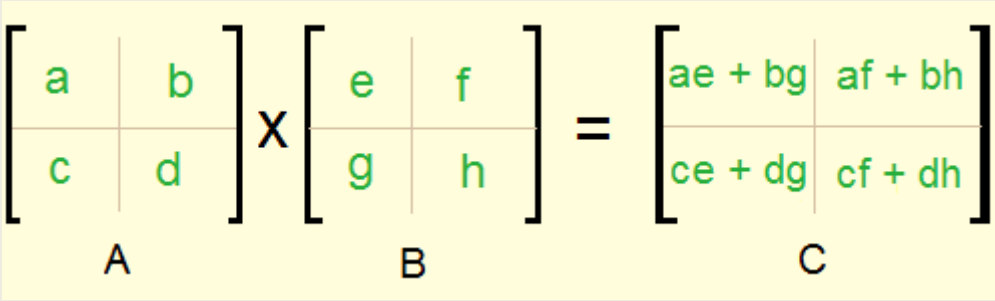
- Complexity:

$O(n \times m)$ *or* $O(n\text{^}2)$

```
for i = 1 to n
    for j = 1 to n
        c_ij = a_ij + b_ij
```

ARRAY

- Multiplication:



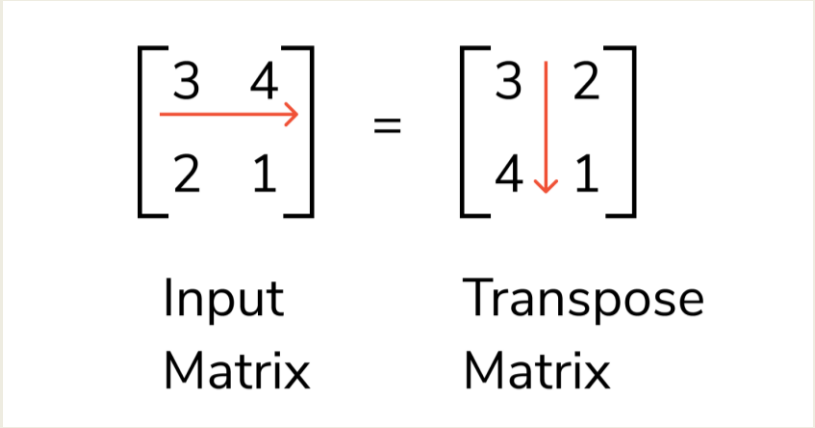
- Complexity:

$O(n \times m \times p)$ *or* $O(n\text{^}3)$

```
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
```

ARRAY

- Transpose:



- Complexity:

$O(n \times m)$ *or* $O(n\text{^}2)$

```
for i = 1 to n
    for j = 1 to n
        b_ji = a_ij
```

MOVING ZEROS

- Challenge: locate all the zeros in a list and push them to the end, leaving the remaining elements in their original order.

```
[8, 0, 3, 0, 12]
[8, 3, 12, 0, 0]
```

```
def move_zeros(a_list):
    zero_index = 0
    for index, n in enumerate(a_list):
        if n != 0:
            a_list[zero_index] = n
            if zero_index != index:
                a_list[index] = 0
            zero_index += 1
    return(a_list)

a_list = [8, 0, 3, 0, 12]
move_zeros(a_list)
print(a_list)
```

COMBINING TWO LISTS

- Challenge:

```
movie_list = [ "Interstellar", "Inception",
               "The Prestige", "Insomnia",
               "Batman Begins"
             ]
```

```
[('Interstellar', 1),
('Inception', 10),
('The Prestige', 10),
('Insomnia', 8),
('Batman Begins', 6)]
```

```
ratings_list = [1, 10, 10, 8, 6]
```

```
movie_list = [ "Interstellar", "Inception",
               "The Prestige", "Insomnia",
               "Batman Begins"
             ]

ratings_list = [1, 10, 10, 8, 6]

print(list(zip(movie_list, ratings_list)))
```

FINDING THE DUPLICATION IN A LIST

- Challenge: check for duplicate items in a list, which you will also have to do often in real- world programming.
  - *Comparing every item to every other item, O(n**2)*
  - *Using Python* ***set****.*

```
a_set = set()
a_set.add("Kanye West")
a_set.add("Kendall Jenner")
a_set.add("Justin Bieber")
print(a_set)

a_set.add('Kanye West')
print(a_set)
```

```
{'Justin Bieber', 'Kanye West', 'Kendall Jenner'}
{'Justin Bieber', 'Kanye West', 'Kendall Jenner'}
```

FINDING THE DUPLICATION IN A LIST

```
def duplicates(an_iterable):
    dups = []
    a_set = set()
    for item in an_iterable:
        l1 = len(a_set)
        a_set.add(item)
        l2 = len(a_set)
        if l1 == l2:
            dups.append(item)
    return dups

a_list = [
    'Susan Adams',
    'Kwame Goodall',
    'Jill Hampton',
    'Susan Adams']

dups = duplicates(a_list)
print(dups)
```

```
['Susan Adams']
```

FINDING THE INTERSECTION OF TWO LISTS

- Challenge:

```
this_weeks_winners = [2, 43, 48, 62, 64, 28, 3]
most_common_winners = [1, 28, 42, 70, 2, 10, 62, 31, 4, 14]
```

```
[2, 62, 28]
```

```
def return_inter(list1, list2):
    list3 = [v for v in list1 if v in list2]
    return list3

list1 = [2, 43, 48, 62, 64, 28, 3]
list2 = [1, 28, 42, 70, 2, 10, 62, 31, 4, 14]
print(return_inter(list1, list2))
```

```
[2, 62, 28]
```

```
def return_inter(list1, list2):
    set1 = set(list1)
    set2 = set(list2)
    return list(set1.intersection(set2))

list1 = [2, 43, 48, 62, 64, 28, 3]
list2 = [1, 28, 42, 70, 2, 10, 62, 31, 4, 14]
new_list = return_inter(list1, list2)
print(new_list)
```

```
[2, 28, 62]
```