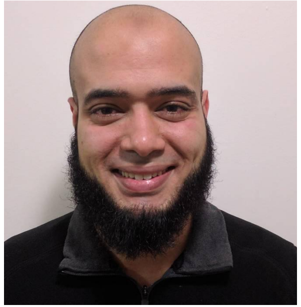
Data Structures

Arrays Homework

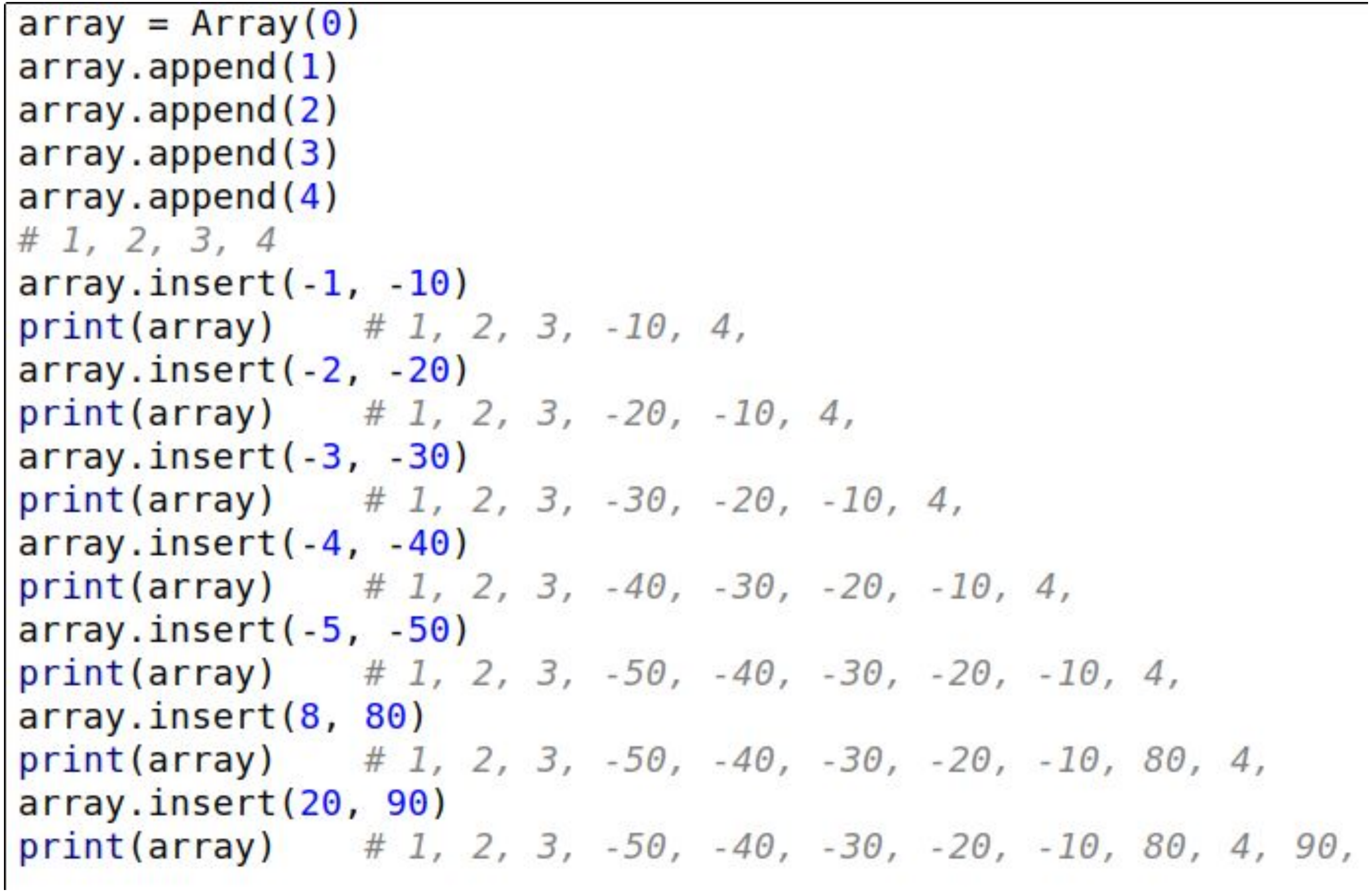
Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Problem #1: Negative indexing

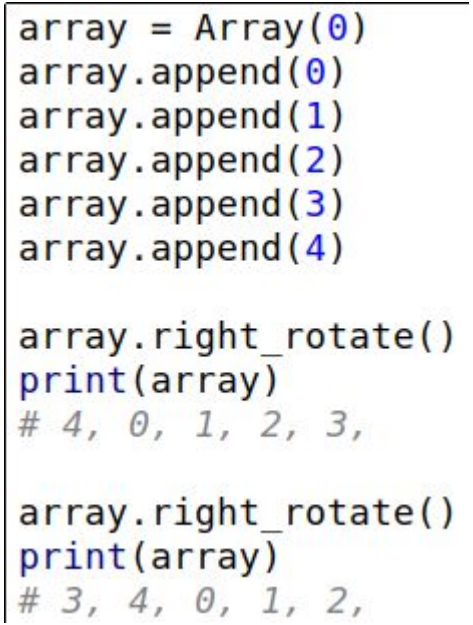
- In the lecture, we implement **insert** function to allow only integer indices in the **range [0, size-1]**
- However, list in Python allows **negative** indexing
- Change the code to allow **negative integer** indexing
  - In practice, the user may also make mistakes by passing a float or a wrong data type
  - Handling these mistakes is out of our scope
- Change your code to work according to the list as follows:



```python
array = Array(0)
array.append(1)
array.append(2)
array.append(3)
array.append(4)
# 1, 2, 3, 4
array.insert(-1, -10)
print(array)    # 1, 2, 3, -10, 4,
array.insert(-2, -20)
print(array)    # 1, 2, 3, -20, -10, 4,
array.insert(-3, -30)
print(array)    # 1, 2, 3, -30, -20, -10, 4,
array.insert(-4, -40)
print(array)    # 1, 2, 3, -40, -30, -20, -10, 4,
array.insert(-5, -50)
print(array)    # 1, 2, 3, -50, -40, -30, -20, -10, 4,
array.insert(8, 80)
print(array)    # 1, 2, 3, -50, -40, -30, -20, -10, 80, 4,
array.insert(20, 90)
print(array)    # 1, 2, 3, -50, -40, -30, -20, -10, 80, 4, 90,
```

Problem #2: Right rotation

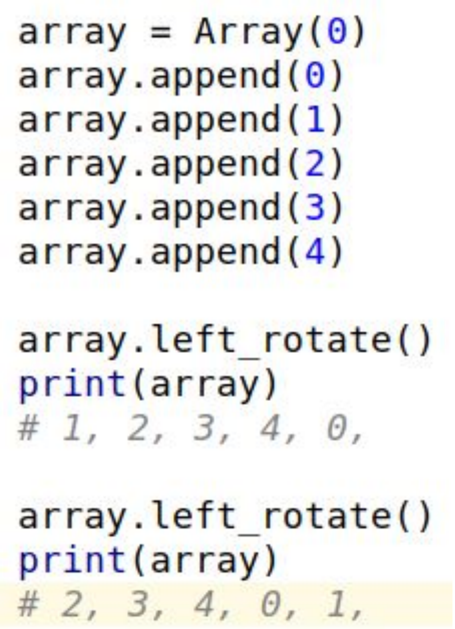
- Consider our Array class.
- Add method: **right_rotate**()
  - The function shifts every element 1 step towards the right.
  - What about the rightmost element? It goes to the first index
- Assume the array content is: 0 1 2 3 4
- After a right rotation it will be: 4 0 1 2 3
  - Notice how, in this case, the '4' has been rotated to **the head of the array!**
- No new array allocation/capacity expansion to occur



```
array = Array(0)
array.append(0)
array.append(1)
array.append(2)
array.append(3)
array.append(4)

array.right_rotate()
print(array)
# 4, 0, 1, 2, 3,

array.right_rotate()
print(array)
# 3, 4, 0, 1, 2,
```

Problem #3: Left rotation

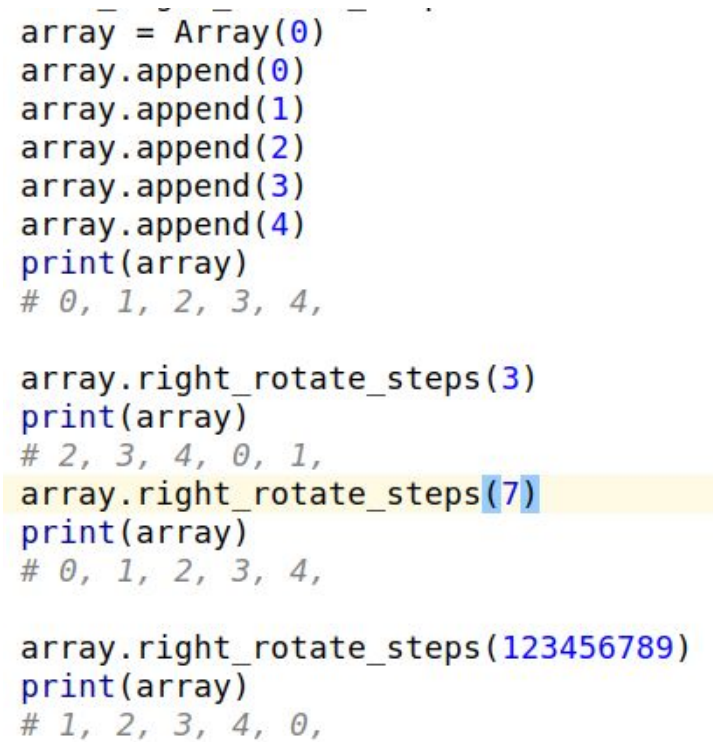
- Add method left_rotate()
  - The function rotates the whole array 1 step to the left
  - However, in this case, the leftmost element will be 'rotated' around to the back of the array!
- Assume the array content is: 0 1 2 3 4
- After a left rotation, it will be: 1 2 3 4 0
  - Notice how the 0 has 'rotated' to the tail of the array after applying left_rotate()



```
array = Array(0)
array.append(0)
array.append(1)
array.append(2)
array.append(3)
array.append(4)

array.left_rotate()
print(array)
# 1, 2, 3, 4, 0,

array.left_rotate()
print(array)
# 2, 3, 4, 0, 1,
```

Problem #4: Right rotation with steps

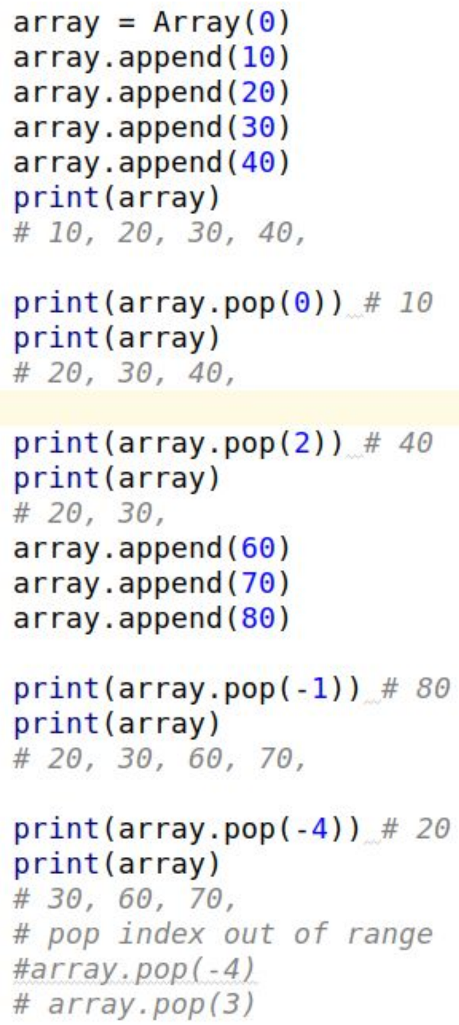
- Implement test_right_rotate_steps(**times**)
- This one applies the right rotation **times** time
- Assume array content is: 0 1 2 3 4
- right_rotate(2) ⇒ it will be: 3 4 0 1 2
- The challenge: times can be up to: 2000000000
    - You code shouldn't be slow



```
array = Array(0)
array.append(0)
array.append(1)
array.append(2)
array.append(3)
array.append(4)
print(array)
# 0, 1, 2, 3, 4,

array.right_rotate_steps(3)
print(array)
# 2, 3, 4, 0, 1,
array.right_rotate_steps(7)
print(array)
# 0, 1, 2, 3, 4,

array.right_rotate_steps(123456789)
print(array)
# 1, 2, 3, 4, 0,
```

Problem #5: pop a position

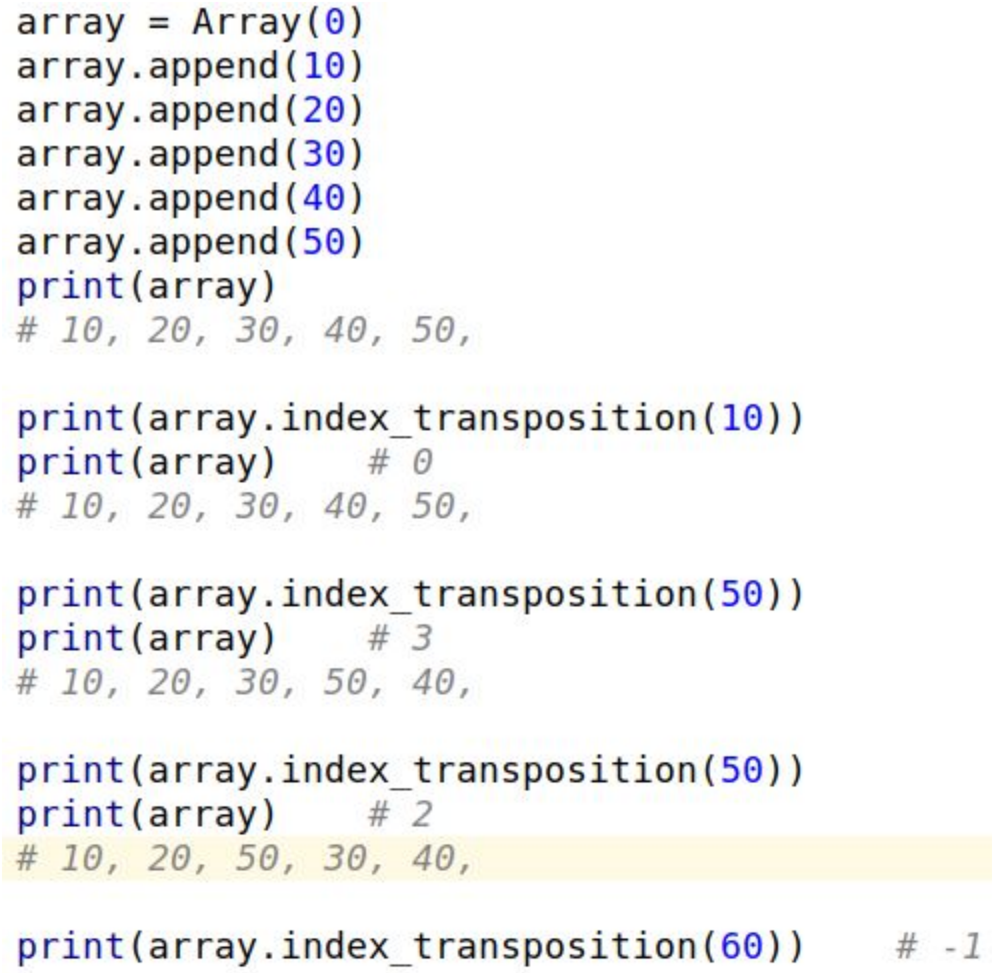
- Implement method pop(idx) to act similar to Python list
    - Index must be in range [-size to size-1], otherwise fail with error msg
        - pop index out of range
        - You can use assertion or throw an exception
- It **returns** the deleted value
- **Remove** this element from the array
- **No new** memory creation
- Code is very **efficient** if the removed element is the **last** one



```
array = Array(0)
array.append(10)
array.append(20)
array.append(30)
array.append(40)
print(array)
# 10, 20, 30, 40,

print(array.pop(0)) # 10
print(array)
# 20, 30, 40,

print(array.pop(2)) # 40
print(array)
# 20, 30,
array.append(60)
array.append(70)
array.append(80)

print(array.pop(-1)) # 80
print(array)
# 20, 30, 60, 70,

print(array.pop(-4)) # 20
print(array)
# 30, 60, 70,
# pop index out of range
#array.pop(-4)
# array.pop(3)
```

Problem #6: Improved search

- Assume our Array is huge and we do many index(value) calls for almost a few small repetitive values
- One way to improve the code speed is: each time you find the value, you **shift** it one step to the **left**
- Eventually, the values that are queried a lot, will move to the head of array
- Implement method: index_transposition(int value)
    - It returns the found position, but consider moving it one step to the left
- Example: 10 20 30 40 50. index_transposition(3)
    - New array 10 30 20 40 50
    - Return 1



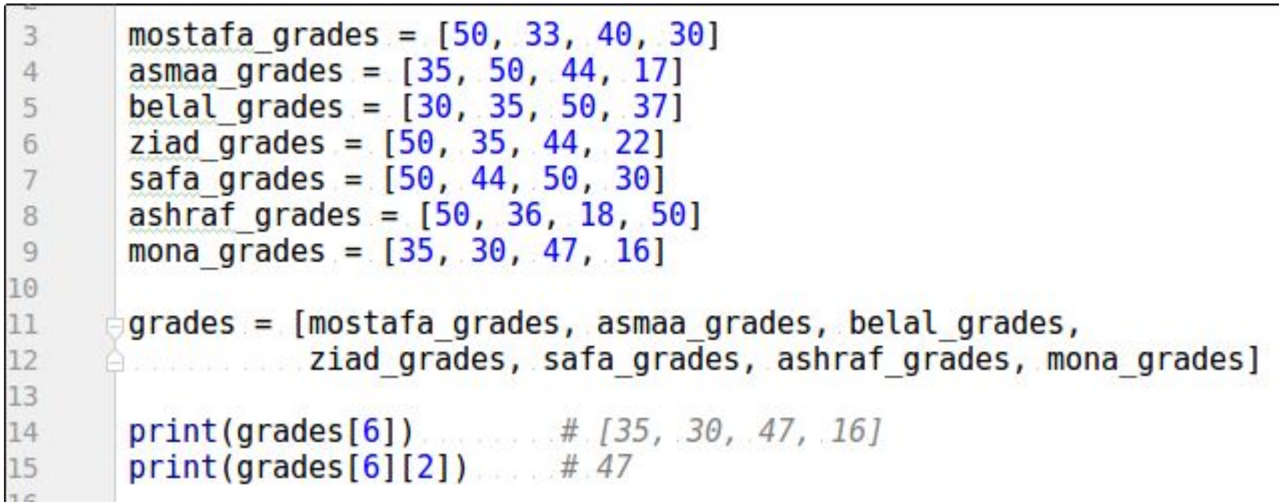
```
array = Array(0)
array.append(10)
array.append(20)
array.append(30)
array.append(40)
array.append(50)
print(array)
# 10, 20, 30, 40, 50,

print(array.index_transposition(10))
print(array)    # 0
# 10, 20, 30, 40, 50,

print(array.index_transposition(50))
print(array)    # 3
# 10, 20, 30, 50, 40,

print(array.index_transposition(50))
print(array)    # 2
# 10, 20, 50, 30, 40,

print(array.index_transposition(60))    # -1
```

Grades for 7 students x 4 subjects

	Math	**Science**	**History**	**Arts**
Mostafa	50	33	40	30
Asmaa	35	50	44	17
Belal	30	35	50	37
Ziad	50	35	44	22
Safa	50	44	50	30
Ashraf	50	36	18	50
Mona	35	30	**47**	16

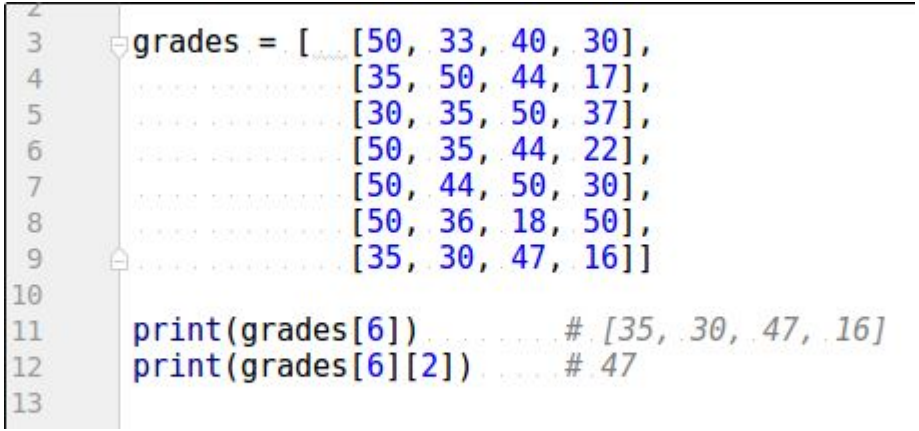
- This is called a matrix/table
  - The blue numbers
- 7 rows
  - Row 0, 1, 2, … 6
  - Row 0 for mostafa
  - Row 6 for mona
- 4 Columns
  - Column 0, 1, 2, 3
  - Column 0 for Math
- Value of table: row 6, col 2
  - 47 (Mona & History)
  - Notation: [6][2]

Creation using Python list: nested list



```python
mostafa_grades = [50, 33, 40, 30]
asmaa_grades = [35, 50, 44, 17]
belal_grades = [30, 35, 50, 37]
ziad_grades = [50, 35, 44, 22]
safa_grades = [50, 44, 50, 30]
ashraf_grades = [50, 36, 18, 50]
mona_grades = [35, 30, 47, 16]

grades = [mostafa_grades, asmaa_grades, belal_grades,
          ziad_grades, safa_grades, ashraf_grades, mona_grades]

print(grades[6])        # [35, 30, 47, 16]
print(grades[6][2])     # 47
```

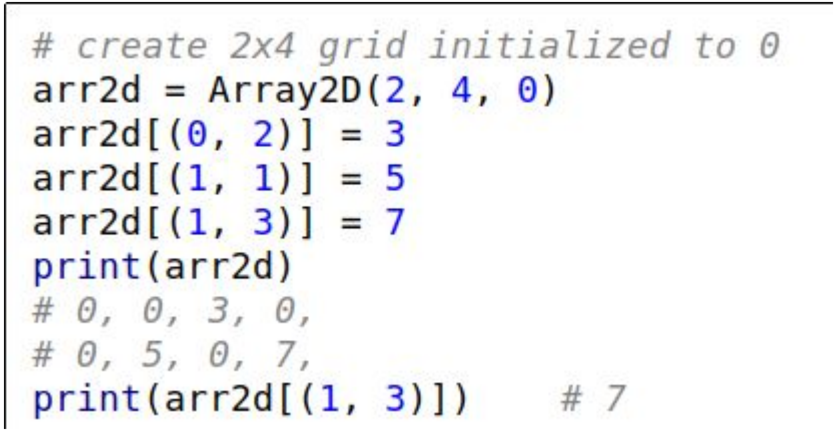
Creation using Python list: nested list



```python
grades = [  [50, 33, 40, 30],
            [35, 50, 44, 17],
            [30, 35, 50, 37],
            [50, 35, 44, 22],
            [50, 44, 50, 30],
            [50, 36, 18, 50],
            [35, 30, 47, 16]]

print(grades[6])        # [35, 30, 47, 16]
print(grades[6][2])     # 47
```

Problem #7: 2D Array

- In this homework, you will create a class Array2D, which will create a FIXED grid of requested rows and columns. The code should utilize the Array class we built before
- For simplicity, only provide this functionalities, assume correct indices
- Optional: Add a lot of matrix functionalities for your class
  - E.g. add / multiply 2 matrices



```
# create 2x4 grid initialized to 0
arr2d = Array2D(2, 4, 0)
arr2d[(0, 2)] = 3
arr2d[(1, 1)] = 5
arr2d[(1, 3)] = 7
print(arr2d)
# 0, 0, 3, 0,
# 0, 5, 0, 7,
print(arr2d[(1, 3)])    # 7
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."