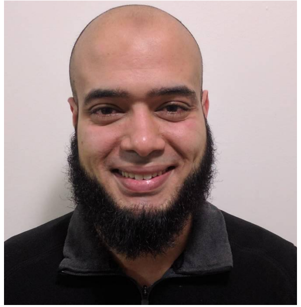
Data Structures

The Array

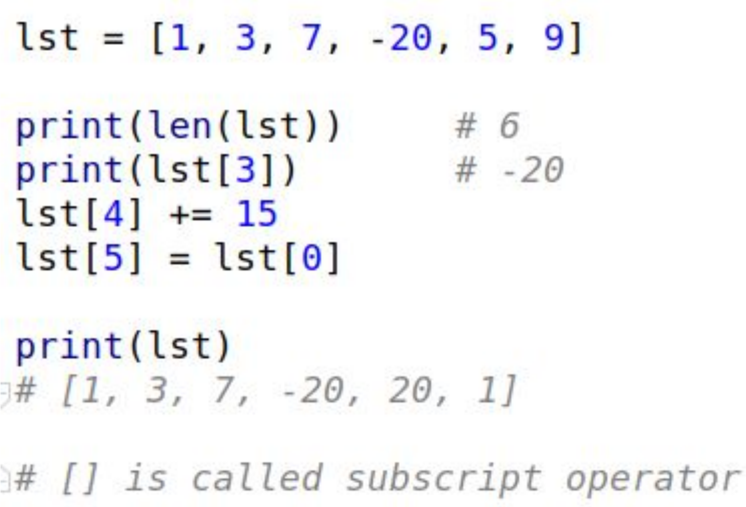
Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

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



Recall the List

- **List** is one of the most commonly used data structures
- Like any data structure, it has **several methods** enabling manipulation of its data
  - Dynamic size, multiple data types, etc
- Without understanding how it is implemented internally, you might be puzzled by its **performance** (e.g. how fast)
  - More formal details in the complexity topic



```
lst = [1, 3, 7, -20, 5, 9]

print(len(lst))      # 6
print(lst[3])        # -20
lst[4] += 15
lst[5] = lst[0]

print(lst)
# [1, 3, 7, -20, 20, 1]

# [] is called subscript operator
```

How is list implemented?

- There are different implementations **for Python**
  - Hence, the performance varies
- **CPython** is the reference implementation of Python
  - The default and **most widely used** implementation
- Written in **C** Programming language

Arrays in C/C++

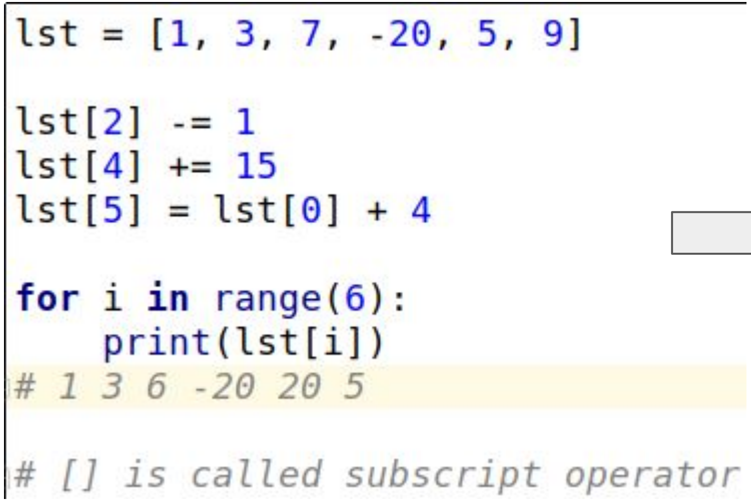
- List is implemented using an **Array**
  - Array initial size is **FIXED**. All elements sre of the **SAME** data type
  - Array of N integers ⇒ N **consecutive** integers in the memory
  - If an integer is represented by 4 bytes and each byte is 8 bits, (in total 32 bits per integer) then an array of 6 integers is **consecutive** 6 * 32 = 192 bits
- The advantage of **consecutive memory**?
  - With simple math formula, we can know where is the location of the **ith integer**

1	3	7	-20	5	9

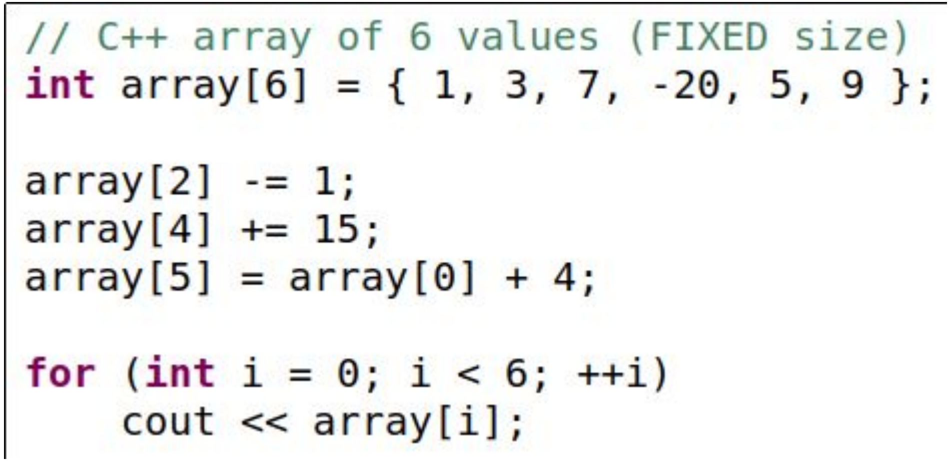
  - If array[0] starts in the memory at location 10000
  - The 4th cell, array[3], is at location: 10000 + 3 * 32 = 10096
- With pointers, we can create/delete arrays content (*as a whole block*)

Behind the scenes

- When you create a list, a pointer to **C array** is used in the memory



```
lst = [1, 3, 7, -20, 5, 9]

lst[2] -= 1
lst[4] += 15
lst[5] = lst[0] + 4

for i in range(6):
    print(lst[i])
# 1 3 6 -20 20 5

# [] is called subscript operator
```



```
// C++ array of 6 values (FIXED size)
int array[6] = { 1, 3, 7, -20, 5, 9 };

array[2] -= 1;
array[4] += 15;
array[5] = array[0] + 4;

for (int i = 0; i < 6; ++i)
    cout << array[i];
```

Array **limitations**

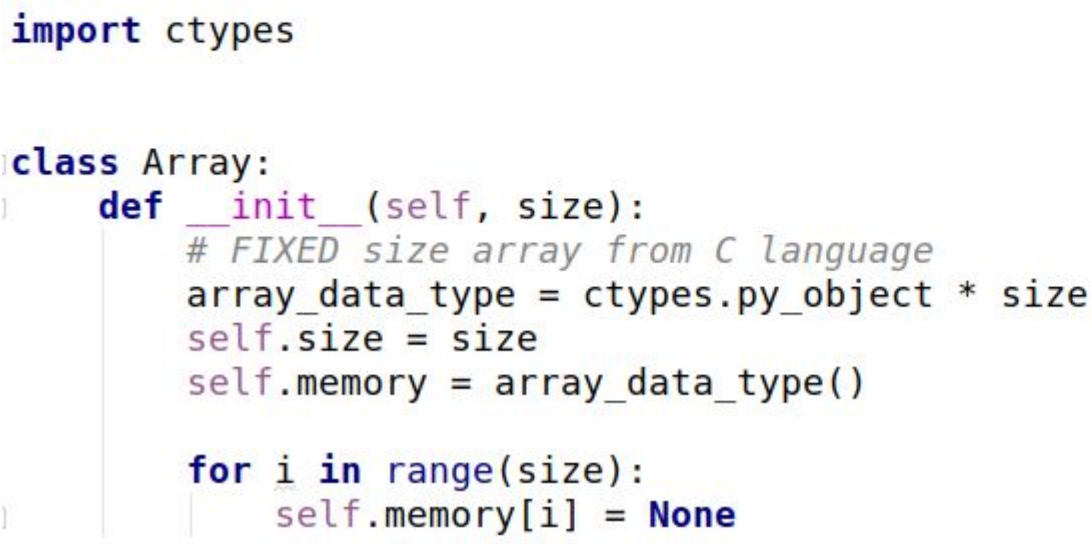
- A C/C++ built-in array is mainly **limited by its initial size**
  - Once size is determined, it is fixed
  - This means no flexibility; i.e. we cannot insert/remove elements that **change** the size
- In practice, we need something that is more **memory flexible**
  - Such as append, extend, pop, remove, delete, which all change the size!
- In addition, we want to support more methods such as min, max, slice, multiply, etc
  - The array allows **only 2 operations**: set and get the elements using the [] operator
- Finally, we need to use **multiple data types**
  - lst = [8, **'ali'**, 4.5, **'D'**]
  - Out of our scope (a matter of implementation rather than performance)

Creating actual C array in Python

- The **ctypes module** allows us to create a C array of FIXED size
  - **import** ctypes
  - array_data_type = ctypes.py_object * 6
  - memory = array_data_type()
- The last 2 lines
  - We created a **data type** of **6 elements**, and each can be a Python object (of any value)
    - This will be a fixed C array.
      - You can’t add more elements.
      - You can’t remove a specific element
  - Then we created an actual object from this class and set a reference in memory
- We will **use this data type** to see how to build our **own list data structure** from a limited data structure!

Our C Array

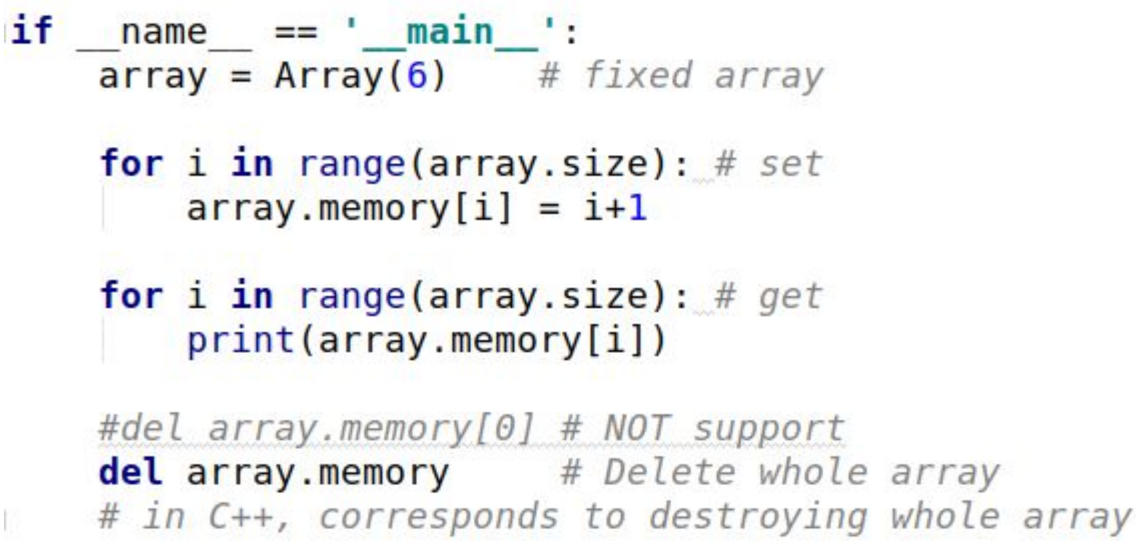
- Let's create a class to help us create objects easily



```python
import ctypes


class Array:
    def __init__(self, size):
        # FIXED size array from C language
        array_data_type = ctypes.py_object * size
        self.size = size
        self.memory = array_data_type()

        for i in range(size):
            self.memory[i] = None
```

Our C Array

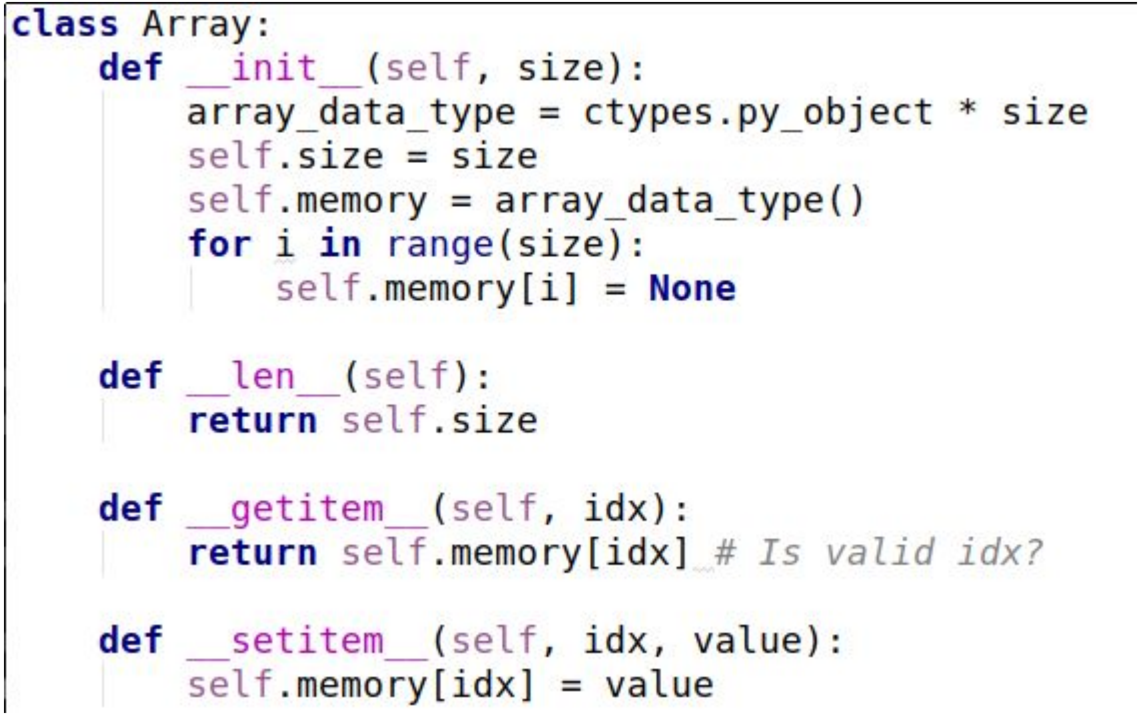
- Now, we can create object of FIXED array size in a normal way
    - Print: 1 2 3 4 5 6



```python
if __name__ == '__main__':
    array = Array(6)    # fixed array

    for i in range(array.size): # set
        array.memory[i] = i+1

    for i in range(array.size): # get
        print(array.memory[i])

    #del array.memory[0] # NOT support
    del array.memory     # Delete whole array
    # in C++, corresponds to destroying whole array
```

Let’s add some **special methods** to our class

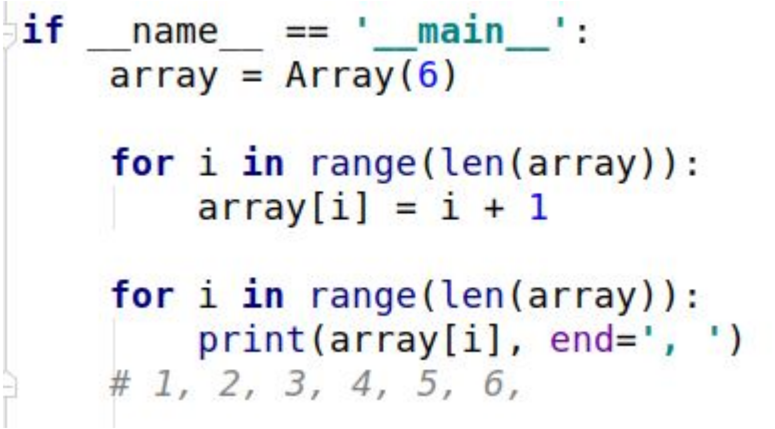
- You learn special methods during **OOP**
- __len__
  - Allows len(array)
- __getitem__
  - Allow print(array[i])
- __setitem__
  - Allow array[i] = i
- We can add **__repr__**
  - To allow printing of the whole object



```
class Array:
    def __init__(self, size):
        array_data_type = ctypes.py_object * size
        self.size = size
        self.memory = array_data_type()
        for i in range(size):
            self.memory[i] = None

    def __len__(self):
        return self.size

    def __getitem__(self, idx):
        return self.memory[idx] # Is valid idx?

    def __setitem__(self, idx, value):
        self.memory[idx] = value
```

Using special methods

- Now, the code looks like a Python list!
- However, it is **limited** in:
  - Appending or removing elements *which affect the used memory size*
  - Supporting functionalities (i.e. min)
- In practice, we need flexible data structures!
- Tip: this limited class is more **memory efficient** than the list class due to its simplicity. There isn't much extra information behind the scenes.



```
if __name__ == '__main__':
    array = Array(6)

    for i in range(len(array)):
        array[i] = i + 1

    for i in range(len(array)):
        print(array[i], end=', ')
    # 1, 2, 3, 4, 5, 6,
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."