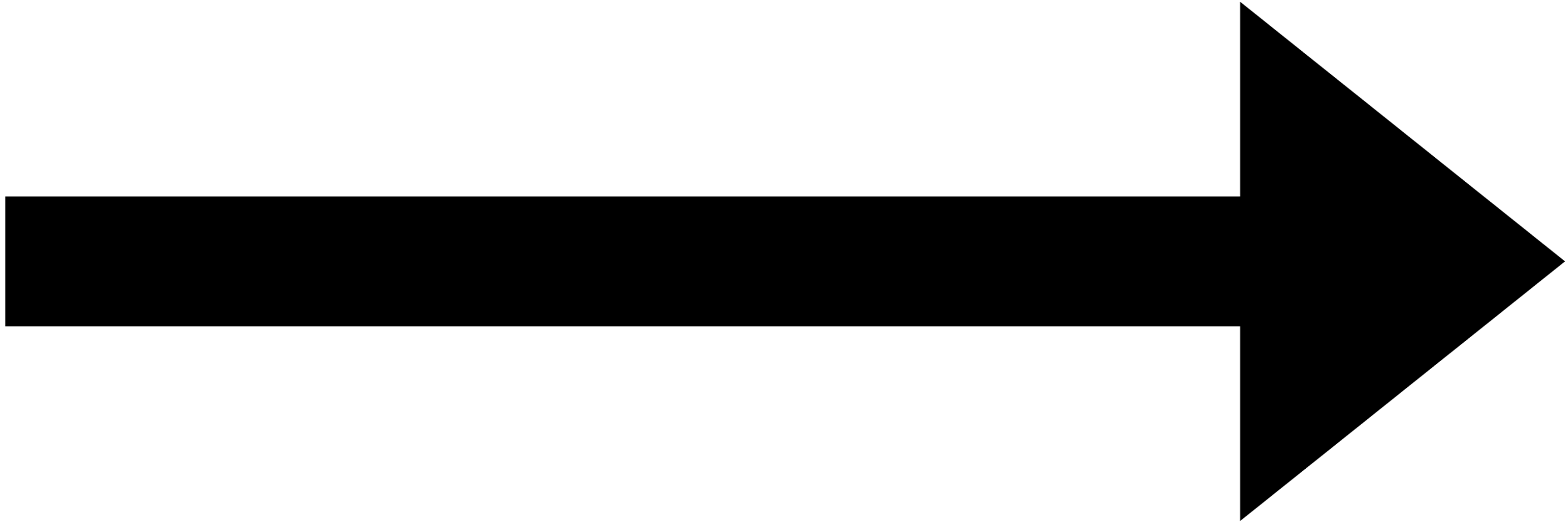
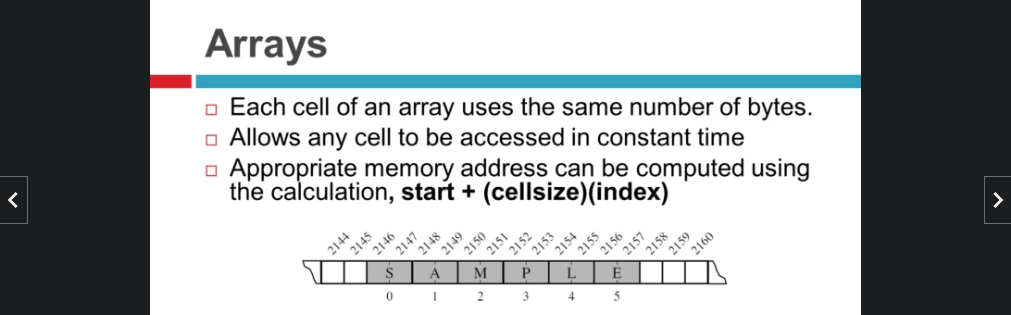
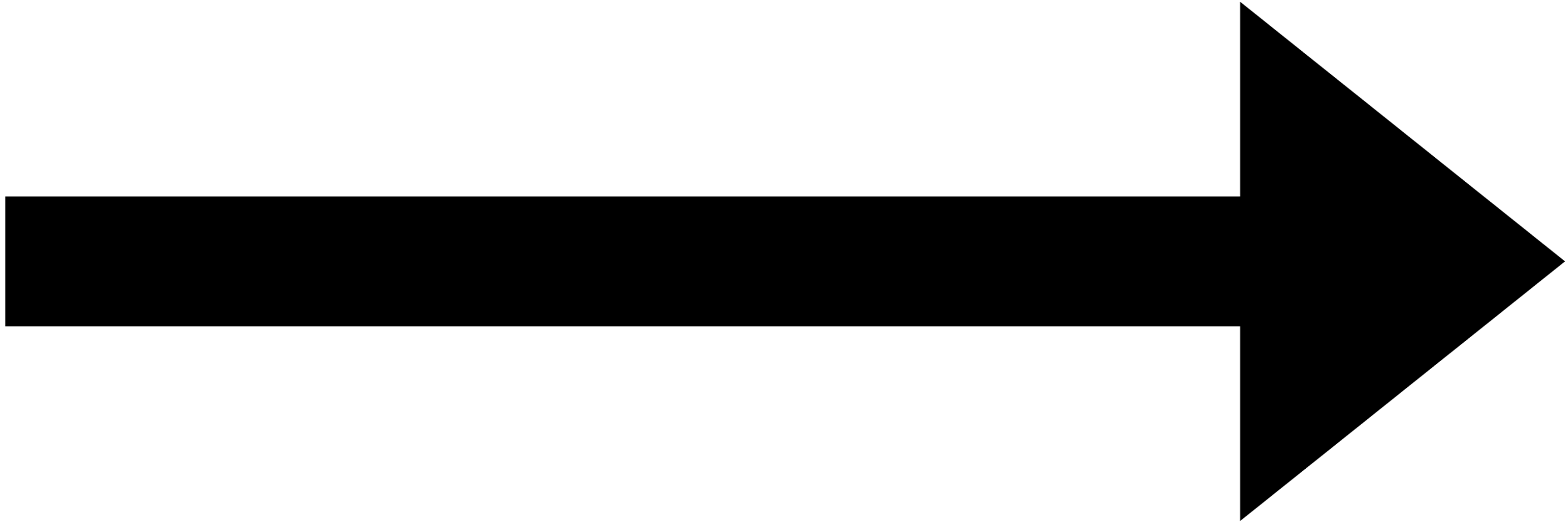
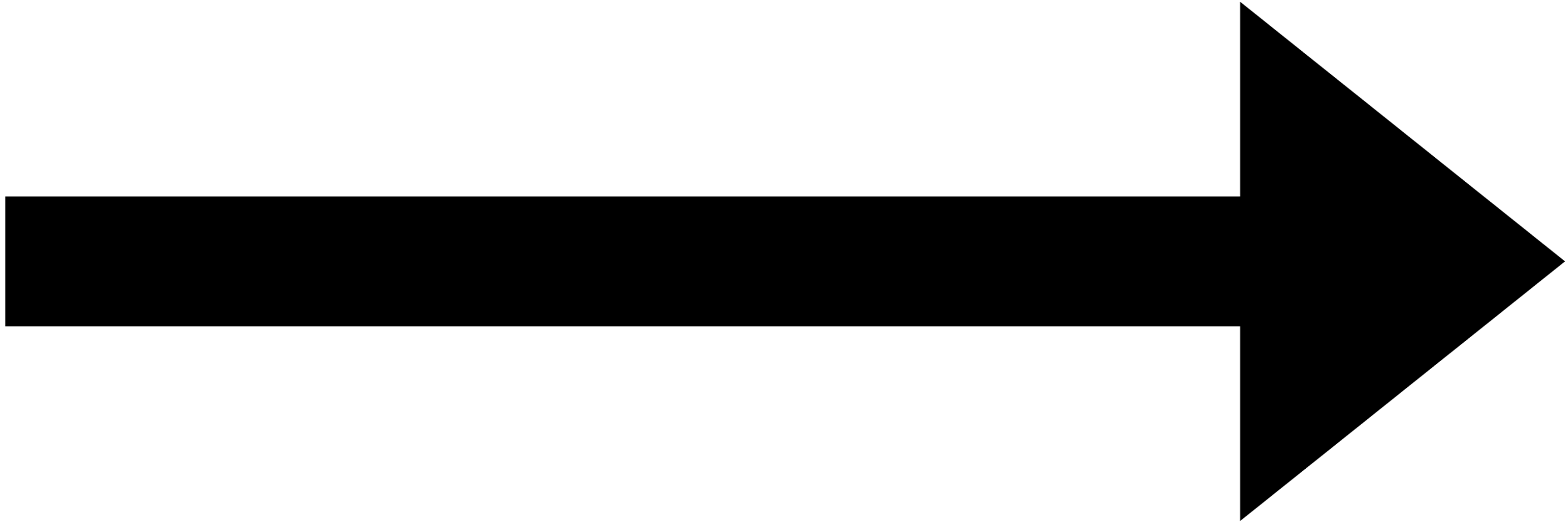
Arrays

An array is a data structure consisting of a collection of same data type elements (values or variables), of same memory size, each identified by at least one array index that is stored at contiguous (next to each other) memory locations.



Types of Array operations:

* **Traversal**: Traverse through the elements of an array.
* **Insertion:** Inserting a new element in an array.
* **Deletion**: Deleting element from the array.
* **Searching**: Search for an element in the array.
* **Sorting**: Maintaining the order of elements in the array.

Advantages of using Arrays:

* Arrays allow random access to elements. This makes accessing elements by position faster.
* Arrays have better cache locality which makes a pretty big difference in performance.
* Arrays represent multiple data items of the same type using a single name.
* Arrays store multiple data of similar types with the same name.
* Array data structures are used to implement the other data structures like linked lists, stacks, queues, trees, graphs, etc.

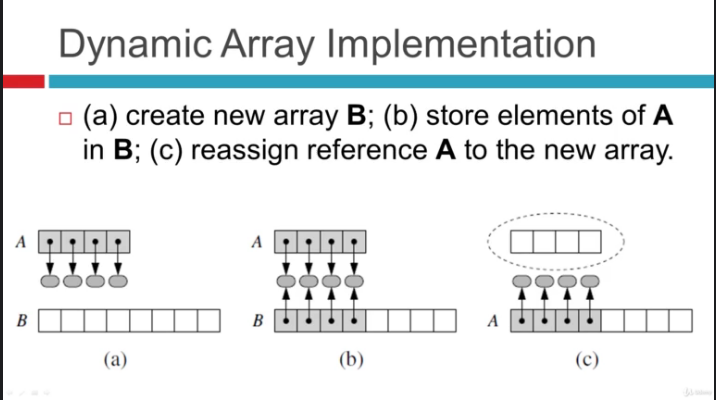
Disadvantages of Array:

* As arrays have a fixed size, once the memory is allocated to them, it cannot be increased or decreased, making it impossible to store extra data if required. An array of fixed size is referred to as a static array.
* Allocating less memory than required to an array leads to loss of data.
* An array is homogeneous in nature so, a single array cannot store values of different data types.
  + Arrays store data in contiguous memory locations, which makes deletion and insertion very difficult to implement. This problem is overcome by implementing linked lists, which allow elements to be accessed sequentially.

Dynamic Array

A dynamic array is array that expands its capacity as you add more elements. So you don't need to determine the size ahead of time.

We cant actually grow the array instead When the arrays capacity is full, we create a new array which will hold the elements of already full array plus an extra amount of free space.

  
  
Dynamic Array Implementation

import ctypes

class DynamicArray(object):

'''

DYNAMIC ARRAY CLASS (Similar to Python List)

'''

def \_\_init\_\_(self):

self.n = 0 # Count actual elements (Default is 0)

self.capacity = 1 # Default Capacity

self.A = self.make\_array(self.capacity)

def \_\_len\_\_(self):

"""

Return number of elements stored in array

"""

return self.n

def \_\_getitem\_\_(self, k):

"""

Return element at index k

"""

if not 0 <= k < self.n:

# Check it k index is in bounds of array

return IndexError('K is out of bounds !')

return self.A[k] # Retrieve from the array at index k

def append(self, ele):

"""

Add element to end of the array

"""

if self.n == self.capacity:

# Double capacity if not enough room

self.\_resize(2 \* self.capacity)

self.A[self.n] = ele # Set self.n index to element

self.n += 1

def insertAt(self, item, index):

"""

This function inserts the item at any specified index.

"""

if index < 0 or index > self.n:

print("please enter appropriate index..")

return

if self.n == self.capacity:

self.\_resize(2\*self.capacity)

for i in range(self.n-1, index-1, -1):

self.A[i+1] = self.A[i]

self.A[index] = item

self.n += 1

def delete(self):

"""

This function deletes item from the end of array

"""

if self.n == 0:

print("Array is empty deletion not Possible")

return

self.A[self.n-1] = 0

self.n -= 1

def removeAt(self, index):

"""

This function deletes item from a specified index..

"""

if self.n == 0:

print("Array is empty deletion not Possible")

return

if index < 0 or index >= self.n:

return IndexError("Index out of bound....deletion not possible")

if index == self.n-1:

self.A[index] = 0

self.n -= 1

return

for i in range(index, self.n-1):

self.A[i] = self.A[i+1]

self.A[self.n-1] = 0

self.n -= 1

def \_resize(self, new\_cap):

"""

Resize internal array to capacity new\_cap

"""

B = self.make\_array(new\_cap) # New bigger array

for k in range(self.n): # Reference all existing values

B[k] = self.A[k]

self.A = B # Call A the new bigger array

self.capacity = new\_cap # Reset the capacity

def make\_array(self, new\_cap):

"""

Returns a new array with new\_cap capacity

"""

return (new\_cap \* ctypes.py\_object)()