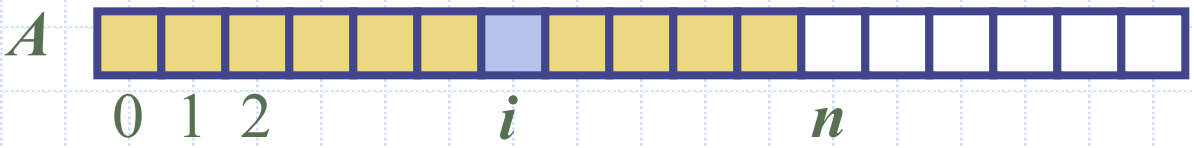
Array-Based Sequences

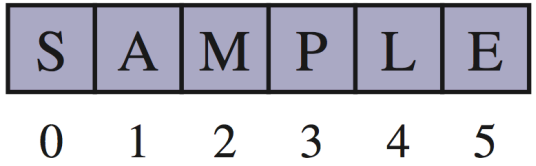
Python Sequence Classes

- Python has built-in types, **list**, **tuple**, and **str**.
- Each of these **sequence** types supports indexing to access an individual element of a sequence, using a syntax such as A[i]
- Each of these types uses an **array** to represent the sequence.
  - An array is a set of memory locations that can be addressed using consecutive indices, which, in Python, start with index 0.

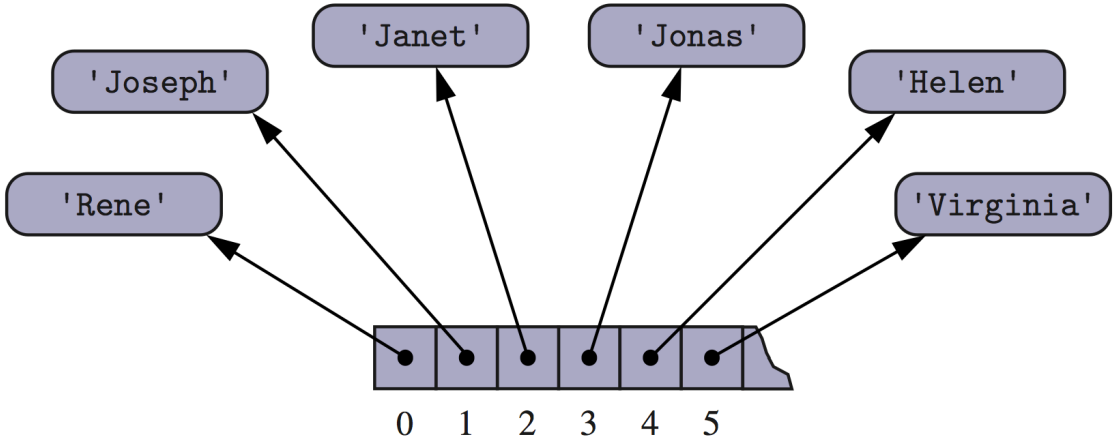


Arrays of Characters or Object References

- An array can store primitive elements, such as characters, giving us a **compact array.**



- An array can also store references to objects.



Compact Arrays

- Primary support for compact arrays is in a module named **array**.
  - That module defines a class, also named array, providing compact storage for arrays of primitive data types.
- The constructor for the array class requires a type code as a first parameter, which is a character that designates the type of data that will be stored in the array.

```
primes = array('i', [2, 3, 5, 7, 11, 13, 17, 19])
```

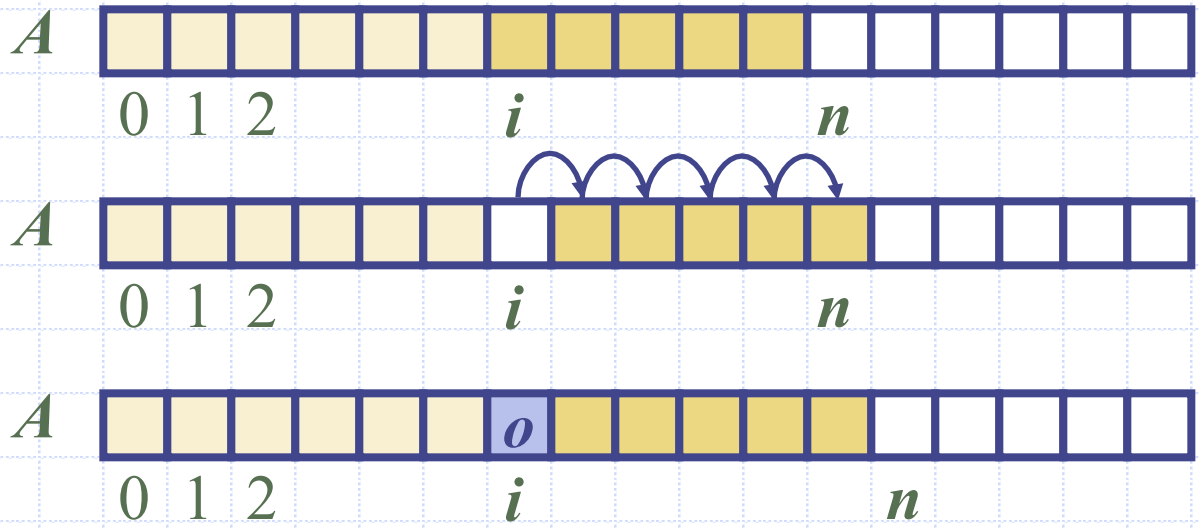
Type Codes in the array Class

- Python's array class has the following type codes:

Code	C Data Type	Typical Number of Bytes
`'b'`	signed char	1
`'B'`	unsigned char	1
`'u'`	Unicode char	2 or 4
`'h'`	signed short int	2
`'H'`	unsigned short int	2
`'i'`	signed int	2 or 4
`'I'`	unsigned int	2 or 4
`'l'`	signed long int	4
`'L'`	unsigned long int	4
`'f'`	float	4
`'d'`	float	8

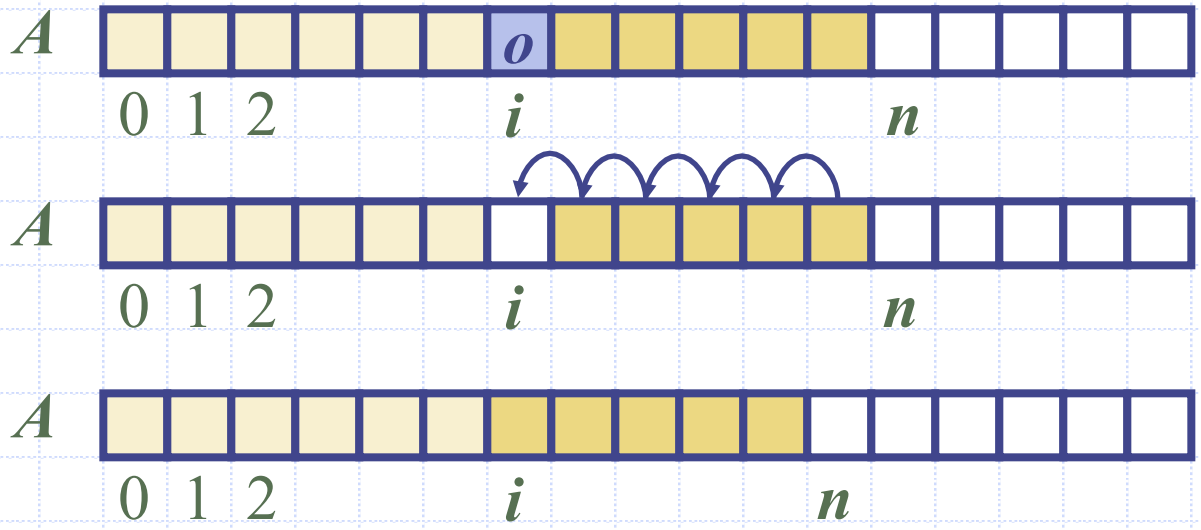
Insertion

- In an operation *insert*(i, o), we need to make room for the new element by shifting forward the $n - i$ elements $A[i], \ldots, A[n-1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



Element Removal

- In an operation ***remove***(i), we need to fill the hole left by the removed element by shifting backward the $n - i - 1$ elements $A[i + 1], \ldots, A[n - 1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



Performance

- In an array based implementation of a dynamic list:
  - The space used by the data structure is $O(n)$
  - Indexing the element at I takes $O(1)$ time
  - ***add*** and ***remove*** run in $O(n)$ time in worst case
- In an ***add*** operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one…

Growable Array-based Array List

- In an add(o) operation (without an index), we could always add at the end: **append(o)**
- When the array is full, we replace the array with a larger one
- How large should the new array be?
  - Incremental strategy: increase the size by a constant c
  - Doubling strategy: double the size

Algorithm *add*(o)
  if $t = S.length - 1$ **then**
    $A \leftarrow$ **new array of size …**
    for $i \leftarrow 0$ **to** $n-1$ **do**
      $A[i] \leftarrow S[i]$
    $S \leftarrow A$
  $n \leftarrow n + 1$
  $S[n-1] \leftarrow o$

Comparison of the Strategies

- We compare the incremental strategy and the doubling strategy by analyzing the total time $T(n)$ needed to perform a series of n add(o) operations
- We assume that we start with an empty stack represented by an array of size 1
- We call amortized time of an add operation the average time taken by an add over the series of operations, i.e., $T(n)/n$

Incremental Strategy Analysis

- We replace the array $k = n/c$ times
- The total time $T(n)$ of a series of n add operations is proportional to

$$n + c + 2c + 3c + 4c + \ldots + kc =$$

$$n + c(1 + 2 + 3 + \ldots + k) =$$

$$n + ck(k + 1)/2$$

- Since c is a constant, $T(n)$ is $O(n + k^2)$, i.e., $O(n^2)$
- The amortized time of an add operation is $O(n)$

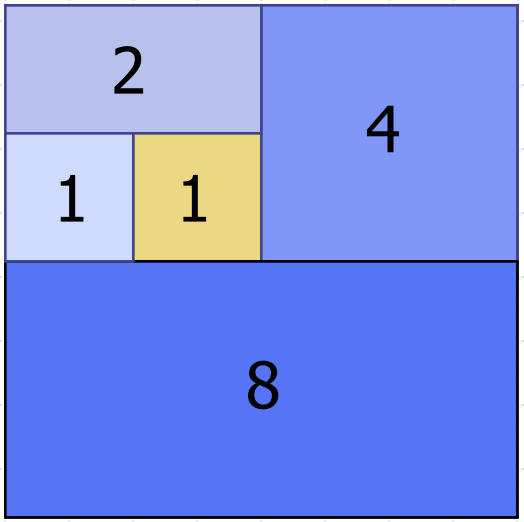
Doubling Strategy Analysis

- We replace the array $k = \log_2 n$ times
- The total time $T(n)$ of a series of n add operations is proportional to

$$n + 1 + 2 + 4 + 8 + \ldots + 2^k =$$
$$n + 2^{k+1} - 1 =$$
$$3n - 1$$

- $T(n)$ is $O(n)$
- The amortized time of an add operation is $O(1)$

geometric series



Python Implementation

- Viết chương trình mô phỏng mảng động. Mảng động được biểu diễn thông qua: kích thước thực tế của mảng, kích thước tối đa và các phần tử được biểu diễn bằng mảng. Mảng động có các chắc năng như tạo mảng, thay đổi kích thước, thêm 1 phần tử vào cuối mảng, thêm 1 phần tử vào 1 vị trí bất kỳ, remove 1 phần tử thông qua giá trị, remove 1 phần tử bất kỳ thông qua vị trí.

Python Implementation

```
1   import ctypes                                       # provides low-level arrays
2   from random import randint
3
4   class DynamicArray:
5     """A dynamic array class akin to a simplified Python list."""
6
7     def __init__(self):
8       """Create an empty array."""
9       self._size = 0                                    # count actual elements
10      self._capacity = 1                                # default array capacity
11      self._element = self._make_array(self._capacity)      # low-level array
12
13    def _make_array(self, c):  # nonpublic utitity
14        """Return new array with capacity c."""
15        return (c * ctypes.py_object)()  # see ctypes documentation
16
17    def showInfor(self):
18      print("Actual element:", self._size)
19      print("Capacity:",self._capacity)
20      for i in range(self._size):
21        print(" ", self._element[i])
22    def __len__(self):
23      """Return number of elements stored in the array."""
24      return self._size
25
26    def __getitem__(self, i):
27      """Return element at index i."""
28      if not 0 <= i < self._size:
29        raise IndexError('invalid index')
30      return self._element[i]                           # retrieve from array
```

Python Implementation

```
31    def _resize(self, c):                          # nonpublic utitity
32      """Resize internal array to capacity c."""
33      b = self._make_array(c)                      # new (bigger) array
34      for k in range(self._size):                     # for each existing value
35        b[k] = self._element[k]
36      self._element = b                                # use the bigger array
37      self._capacity = c
38
39    def append(self, obj):
40      """Add object to end of the array."""
41      if self._size == self._capacity:                # not enough room
42        self._resize(2 * self._capacity)             # so double capacity
43      self._element[self._size] = obj
44      self._size += 1
45
46    def insert(self, index, value):
47      """Insert value at index k, shifting subsequent values rightward."""
48      # (for simplicity, we assume 0 <= k <= n in this verion)
49      if self._size == self._capacity:                # not enough room
50        self._resize(2 * self._capacity)             # so double capacity
51      for j in range(self._size, index, -1):              # shift rightmost first
52        self._element[j] = self._element[j - 1]
53      self._element[index] = value                          # store newest element
54      self._size += 1
55
```

Python Implementation

```
56      def removeByValue(self, value):
57        """Remove first occurrence of value (or raise ValueError)."""
58        # note: we do not consider shrinking the dynamic array in this version
59        for k in range(self._size):
60          if self._element[k] == value:               # found a match!
61            for j in range(k, self._size - 1):    # shift others to fill gap
62              self._element[j] = self._element[j + 1]
63            self._element[self._size - 1] = None         # help garbage collection
64            self._size -= 1                           # we have one less item
65            return                                  # exit immediately
66        raise ValueError('value not found')     # only reached if no match
67      def removeByIndex(self, index):
68        """Remove base index"""
69        if index< self._size:
70          for j in range(index, self._size - 1):    # shift others to fill gap
71            self._element[j] = self._element[j + 1]
72          self._element[self._size - 1] = None         # help garbage collection
73          self._size -= 1                           # we have one less item
74        else:
75          print("Out of index")
76    obj= DynamicArray()
77    obj.showInfor()
78    for i in range(10):
79      obj.append(randint(0,10))
80    obj.showInfor()
81    obj.insert(1,33)
82    obj.showInfor()
```