Arrays

Not Lists

Haitham A. El-Ghareeb

October 21, 2019

Faculty of Computers and Information Sciences
Mansoura University
Egypt
helghareeb@mans.edu.eg

Contacts

- https://www.haitham.ws
- https://youtube.com/helghareeb
- https://www.github.com/helghareeb
- http://eg.linkedin.com/in/helghareeb
- helghareeb@mans.edu.eg

Array Structure

The Array Structure

An **array** is the most basic type of container

- Implemented at the hardware level
- Most languages provide arrays as a primitive type
- Can be used with a wide range of problems

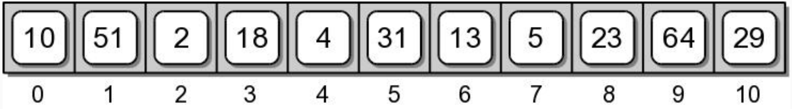
Array Structure

1-D Array

1-D Array

A sequence Structure

- Composed of multiple elements
- Elements are stored in contiguous bytes of memory
- Entire contents is known by a single name
- Individual elements can be accessed by subscript



Array vs. Python List

Both are sequences, but there are two major differences

- Arrays only have 3 operations
  - array creation
  - reading a specific element
  - writing a specific element
- The size of an array is fixed

Why Study Arrays?

- Python provides the list structure as its mutable sequence type
- Do we really need arrays?
    - Many languages only provide the array structure
    - Both structures have their uses

When to use Arrays?

Arrays are best suited to problems where

- maximum number of elements is known upfront
  - array size is fixed
  - the list has extra space that can be wasteful
- only a limited number of operations are needed
  - arrays have 3 operations
  - the list ca manage the items in the container

Array Structure

1-D Array ADT and Implementation

1-D Array ADT

A 1-D array is a collection of contiguous elements with each element identified by integer subscript

- Subscripts start at 0
- Once created, array size can not be changed

```
Array(size)
length()
get_item(index)
set_item(index)
clear(value)
iterator()
```

Array Example 01

- *array_ex_01.py*

Array Example 02

- *array_ex_02.py*

Array Implementation

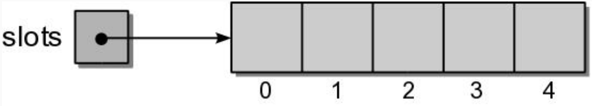
- Python is built using the C language
  - High-level compiled language
  - Provides syntax for working with the hardware
- Python provides the `ctypes` module
  - Access to C data types and functionality
  - Provides for hardware-supported arrays
  - Requires knowledge of C language
  - Not meant for direct use in programs

Hardware Array: Creation

Create a hardware array

```
import ctypes
array_type = ctypes.py_object * 5
slots = array_type()
```

- fixed size
- each element stores a reference to an object

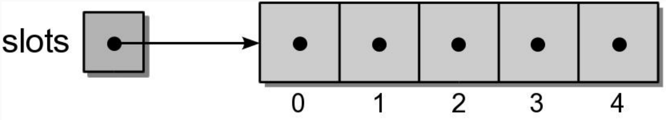


Hardware Array: Initialize

A hardware array has to be intialized before it can be used

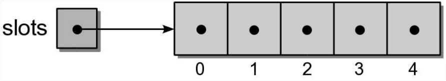
```
for i in range(5):
    slots[i] = None
```

- elements are like any other variable
- we must keep track of the size of the array

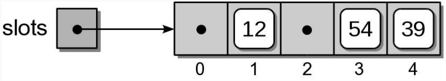


Hardware Array: Add

- References can be stored in any array element

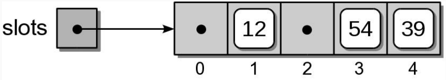



```
slots[1] = 12
slots[3] = 54
slots[4] = 37
```

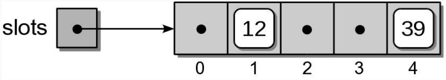


Hardware Array: Remove

- Items can be removed from the array




```
slots[3] = None
```



Array ADT Implementation

- *my_array.py*

Python List

The Python List

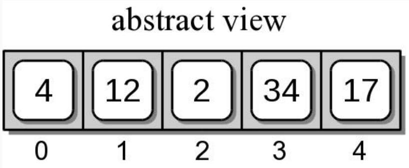
A mutable sequence type container

- Provides operations for managing the collection
- Can grow and / or shrink as needed
- Implemented using an array

List: Construction

- Python list interface provides an abstraction to the actual underlying implementation

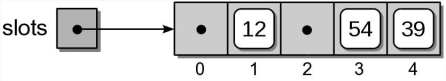
```
py_list = [4,12,2,34,17]
```



List: Implementation

An array is used to store the items of the list

- Created bigger than needed
- Has capacity for future items
- **subarray** - the items are stored in a contiguous subset of the array

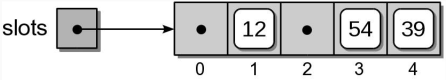


List: Appending an Item

- New items can be added at the end of the list

```
py_list.append(50)
```

- When space is available, the item is stored in the next slot

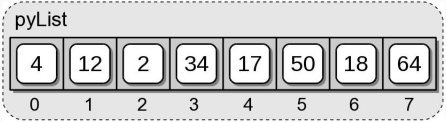


List: Appending an Item

- What happens when the array becomes full?

```
py_list.append(18)
py_list.append(64)
py_list.append(6)
```

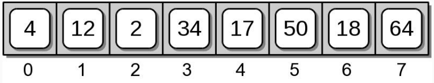
- There is no space for value 6



List: Appending an Item

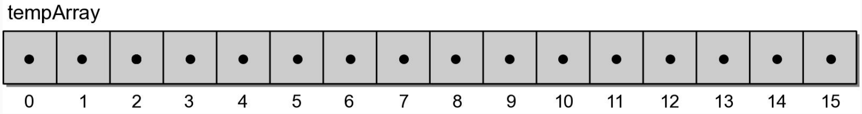
The array has to be expanded

- Can not change the size of an array
- Will require multiple steps

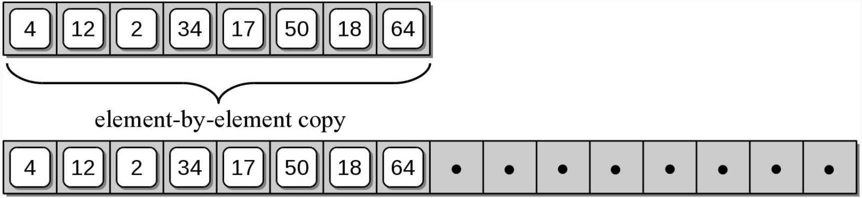


The Original Array

Expanding the Array - 01

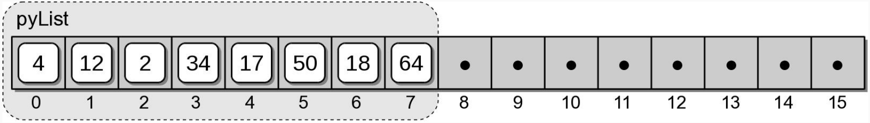


Step 1: create a new array, double the size.

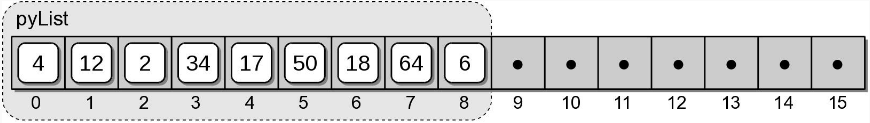


Step 2: copy the items from original array to the new array.

Expanding the Array - 02



Step 3: replace the original array with the new array.

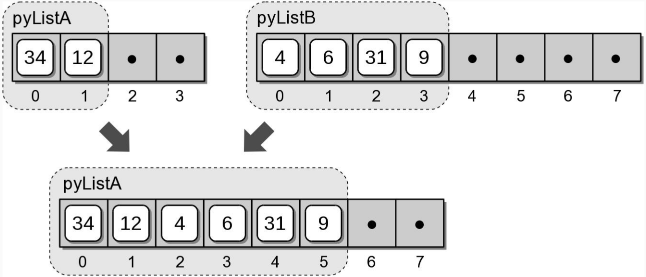


Step 4: store value 6 in the next slot of the new array.

List: Extending

The entire contents of a list can be appended to a second list.

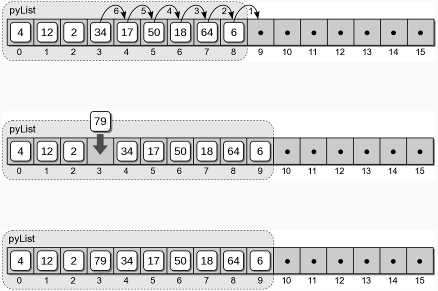
```
py_list_a = [34, 12]
py_list_b = [4, 6, 31, 9]
py_list_a.extend( py_list_b )
```



List: Inserting Items

An item can be iserted anywhere within the list

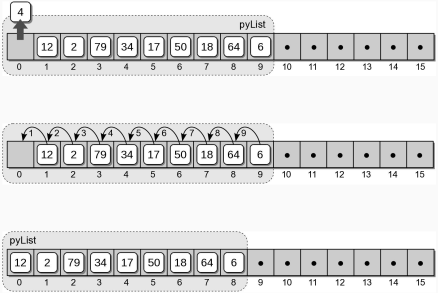
```
py_list.insert(3, 79)
```



List: Removing Items

An item can be removed from position of the list

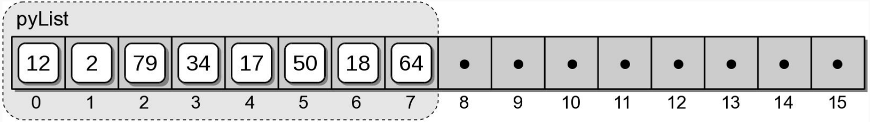
```
py_list.pop(0)
```



List: Removing Items

Removing the last item in the list

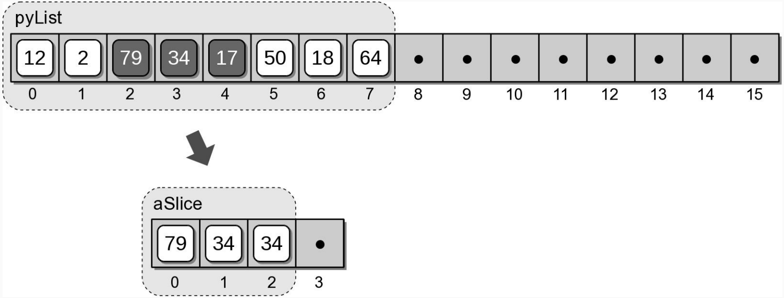
```
py_list.pop()
```



List:Slices

Slicing a list creates a new list from a contiguous subset of elements

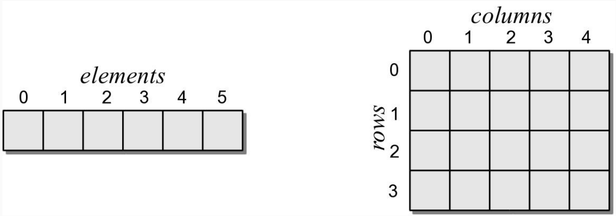
```
a_slice = py_list[2:5]
```



2-D Arrays

2-D Arrays

- Arrays can be defined with multiple dimensions
- Two-dimensional arrays:
  - organize the data in rows and columns
  - element access: [i, j]



2-D Arrays

Arrays of 2 or more dimensions are not supported at the hardware level

- Most languages provide some mechanism for creating and managing multi-dimensional arrays
- 2-D arrays are very common in Computer Science

2-D Arrays

2-D Array ADT and Implementation

2-D Array ADT

A 2-D *array* consists of a collection of elements organized into rows and columns

- Elements are referenced by row and column subscript(start at 0)
- Once created, array size can not be changed

```
Array2D(n_rows, n_cols)
num_rows()
num_cols()
clear(value)
get_item(i,j)
set_item(i,j,value)
```

2-D Array Example

Suppose we have a text file *grades.txt* containing exam grades for multiple students

- Extract the grades from the file
- Store them in a 2-D array
- Compute the average exam grades
- *2d_array_ex_01.py*

2-D Array Example

The contents of the 2-D array produced by the previous code segment

	0	1	2
0	90	96	92
1	85	91	89
2	82	73	84
3	69	82	86
4	95	88	91
5	78	64	84
6	92	85	89

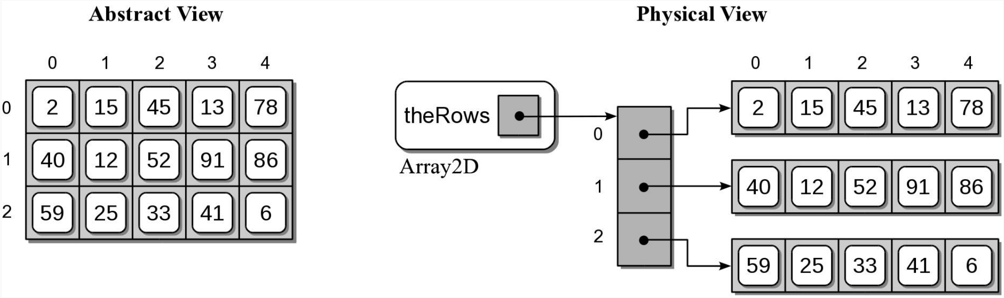
Implementing the 2-D Array

There are various approaches that can be used to implement a 2-D array

- Use a single 1-D array with the elements arranged by row or column
- Use a 1-D array of 1-D arrays

Array of Arrays Implementation

- Each row is stored within its own 1-D array
- A 1-D array is used to store references to each row array



2-D Array Implementation - Common Error

- *array_two_d_err.py*

2-D Array Implementation

- Subscript notation:

```
y = x[r,c]
x[r,c] = z
```

- Subscripts are passed to the methods as tuple
- Must verify the size of the tuple

The Matrix

The Matrix

Matrices are very common in mathematics

- Used for solving systems of linear equations
- Linear Algebra and Computer Graphics

$$\begin{bmatrix} 8 & 12 & 9 & -2 & 3 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \qquad \begin{bmatrix} 5 & -1 \\ 2 & 12 \\ 9 & 10 \\ 8 & 23 \end{bmatrix}$$

The Matrix

The Matrix ADT and Implementation

The Matrix ADT

A matrix is a collection of scalar values arranged in rows and columns as a fixed sized rectangular grid

- Elements are accessed by (row, col) subscript
- Indices start at 0

The Matrix ADT Specifications

```
Matrix(n_rows, n_cols)
num_rows()
num_cols()
get_item(i,j)
set_item(i,j,value)
scale_by(value)
transopose()
add(Matrix)
subtract(Matrix)
multiply(Matrix)
```

The Matrix

Scaling the Matrix

Matrix: Scaling

Modify each element by a common scale factor

- factor < 1 reduces each element value
- factor > 1 increases each element value

$$5\begin{bmatrix} 6 & 7 \\ 8 & 9 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 5*6 & 5*7 \\ 5*8 & 5*9 \\ 5*1 & 5*0 \end{bmatrix} = \begin{bmatrix} 30 & 35 \\ 40 & 45 \\ 5 & 0 \end{bmatrix}$$

The Matrix

Transpose the Matrix

Matrix: Transpose

Swaps the rows and columns of an m x n matrix to create a new n x m matrix

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

The Matrix

Multiply the Matrix

- A matrix of size $m * n$ can be multiplied by a matrix of size $n * p$
- Result is a new $m * p$ matrix

$$\underset{3 \times 2}{\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}} * \underset{2 \times 4}{\begin{bmatrix} 6 & 7 & 8 & 2 \\ 9 & 1 & 0 & 4 \end{bmatrix}} = \underset{3 \times 4}{\begin{bmatrix} 9 & 1 & 0 & 4 \\ 39 & 17 & 16 & 28 \\ 69 & 33 & 32 & 28 \end{bmatrix}}$$

Matrix: Multiplication

- Each element of the new matrix is the result of:
  - summing the product of a row in the lhs matrix
  - by a column in the rhs matrix

$$\begin{bmatrix} \boxed{0 \quad 1} \\ 2 \quad 3 \\ 4 \quad 5 \end{bmatrix} * \begin{bmatrix} \boxed{\begin{matrix}6\\9\end{matrix}} & \begin{matrix}7\\1\end{matrix} & \begin{matrix}8\\0\end{matrix} & \begin{matrix}2\\4\end{matrix} \end{bmatrix} = \begin{bmatrix} \boxed{0*6+1*9} & 0*7+1*1 & 0*8+1*0 & 0*2+1*4 \\ 2*6+3*9 & 2*7+3*1 & 2*8+3*0 & 2*2+3*4 \\ 4*6+5*9 & 4*7+5*1 & 4*8+5*0 & 4*2+5*4 \end{bmatrix}$$

The Matrix

Matrix Implementation

Matrix Implementation

- How should we implement the Matrix ADT?
    - There are different ways to organize the data contained in a matrix
    - Most obvious approach is to use a 2-D array

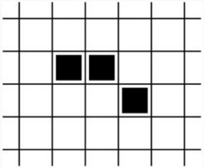
Matrix Implementation

matrix.py

The Game of Life

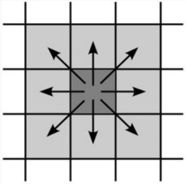
The Game of Life

- Devised by British mathematician John H. Conway; first introduced by Martin Gardner
  - Example of a problem from cellular automata
  - A zero-player solitaire type game
  - Uses an infinite-sized grid of cells:
    - contains an organism (alive)
    - empty (dead)



Life: Cell Neighbors

- Played over a specific period of time:
  - each turn creates a new "generation"
  - based on the current "configuration"
  - next generation determined by applying 4 rules
- Neighbors of a cell



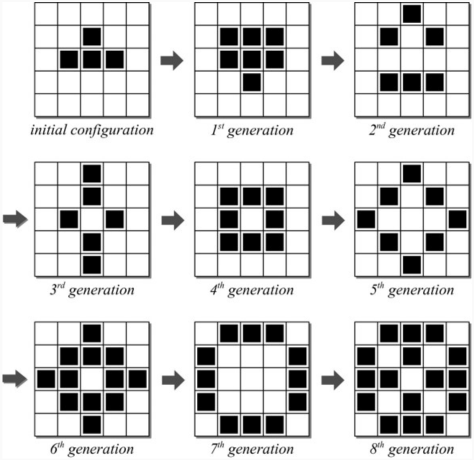
Rules of the Game

1. **Rule 1** if a cell is alive and has either two or three neighbors, the cell remains alive
2. **Rule 2** A living cell with 0 or 1 live neighbors dies from isolation
3. **Rule 3** A living cell with 4 or more live neighbors dies from overpopulation
4. **Rule 4** A dead cell with 3 live neighbors becomes alive resulting from a birth

The Game of Life

Game of Life: Example

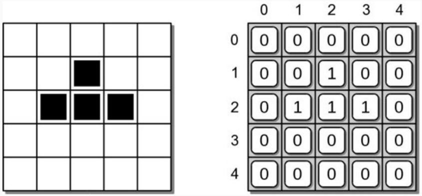
Example of Game



The Game of Life

Game of Life: Implementation

Game Grid Implementation



Summary

Summary

- Arrays
- Python Lists
- 2D Arrays
- Matrices
- Game of Life
- NumPy

https://www.haitham.ws