

# OUTLINE

- 1 CHAPTER 3: CONTAINER CLASSES
  - Python List Implementation
  - Python Dictionaries

# ARRAY-BASED LISTS

## PYTHON'S LIST IMPLEMENTATION

- Computer memory is a sequence of storage locations
- Each storage location has an **address** (i.e. a number) associated with it
- A single data item may be stored across a number of contiguous memory locations
- To retrieve an item from memory we need a way to either **look up** or **compute** the starting address of the object
- To store a collection of objects, we need to have a systematic method for figuring out where each object in the collection is located.

# ARRAY-BASED LISTS

## ARRAYS

- An **array** is a collection of adjacent memory objects all of the same size (a simple method for storing a collection is to allocate a single contiguous area of memory) .
- Usually, the objects in an array are all of the same class. (We then say that the array is **homogeneous**)
- Each object has an **index** giving its position in the array. The first item has index zero, the next item has index one, and so on.
- The values can be accessed efficiently by index.

## ARRAY-BASED LISTS

## ARRAYS

**Example:** Imagine having an array of integers (each having size four bytes) stored at address 1024.

The first item (with index 0) is stored at locations 1024-1027, the next item is stored at locations 1028-1031.

The item with index  $i$  is at address  $1024 + 4 \times i$

- quick to compute whenever the value of item  $i$  must be accessed or changed (in constant time, or  $\Theta(1)$ ).

# ARRAY-BASED LISTS

## PROS OF ARRAYS

- arrays are very memory efficient
- arrays support quick random access  
(i.e. we can "jump" directly to the item we want)

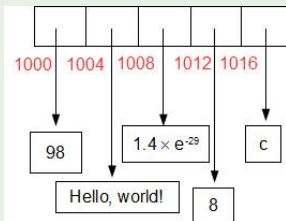
## DIFFICULTIES WITH ARRAYS

- **Heterogeneous data** cannot be kept in an array, since items may have different sizes, with no easy formula to find items by index efficiently.
- **Adding to a Full Array** If the block allocated for an array has filled up, appending new items is not easy. The adjacent space after it might be unavailable, holding other information. (arrays are said to be **static**)

## ARRAY-BASED LISTS

## PYTHON OVERCOMES THESE DIFFICULTIES

- In spite of these difficulties, Python overcomes these problems, and uses **arrays** to implement **list** objects.
- In Python, lists are arrays of **references**, which are **memory addresses** of the actual data objects in scattered locations in memory. Thus heterogeneous data can be handled, since the addresses themselves are all the same size (32 or 64 bits).



# ARRAY-BASED LISTS

## PYTHON OVERCOMES THESE DIFFICULTIES

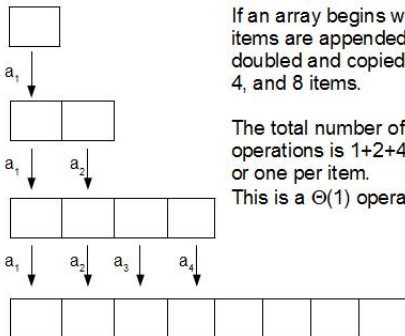
### Adding to a Full Array:

In Python, if a list needs more space, a larger array is allocated, and the old array is copied into it with room to spare for appending new items.

## EFFICIENCY ANALYSIS

## LIST OPERATIONS

- **append** By doubling the size of an array each time it gets full, the average cost of appending a single item is a constant amount.



If an array begins with size 1, and 15 items are appended, the array gets doubled and copied when it has 1, 2, 4, and 8 items.

The total number of copying operations is  $1+2+4+8 = 15$ , or one per item.

This is a  $\Theta(1)$  operation.



## EFFICIENCY ANALYSIS

## LIST OPERATIONS

- **insert** To insert a single item into an array, all the items after it must be copied into the next higher locations.  
On average, this takes  $\frac{1}{2}n$  operations, which is  $\Theta(n)$ .
- **delete** Similarly, to delete a single item from the middle of an array, all the items after it must be copied into the next lower locations.  
This is also  $\Theta(n)$ .

# A DICTIONARY ADT

## A DICTIONARY IS LIKE A LIST

- You can access an item of a **list** by supplying its **index**:  
`l[10]`, `l[1]`, `l[4]` give the values at position 10, 1, and 4 respectively of list `l`.  
There is a function (mapping) from indexes to item values, where integers are the indexes.
- We think of each integer index as a **key** to its data.
- A Python **Dictionary** lets you use a value of *any immutable type* as a valid key.

# A DICTIONARY ADT

## DICTIONARY OPERATIONS

- **Create** Returns an empty dictionary.
- **put(key, value)** Associates the value `value` with `key` in the dictionary.
- **get(key)**  
pre: There is an `X` such that `(key, X)` is in the dictionary.  
post: Returns `X`.
- **delete(key)**  
pre: There is an `X` such that `(key, X)` is in the dictionary.  
post: `(key, X)` is removed from the dictionary.

## PYTHON DICTIONARIES

## EXAMPLE: A SUITS DICTIONARY

```
>>> suits = {"c": "Clubs", "d": "Diamonds",  
            "h": "Hearts", "s": "Spades"}  
>>> suits.get("c")  
'Clubs'  
>>> suits["c"]  
'Clubs'  
>>> suits["j"]  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
KeyError: 'j'
```

# DICTIONARY IMPLEMENTATION

## HASH TABLES

**Hash tables** are the structures used to implement Dictionaries.

They use a function (called a **hash function** - **the heart** of the hash table) which quickly ( $\Theta(1)$ ) computes the index into an array from a key value (which may be a text string, for example).

The values of the dictionary are items referenced in that array, so they can be found in constant time from their keys, regardless of the size of the dictionary.

We will cover hash tables in detail later this semester.

# EXAMPLE: A MARKOV CHAIN LANGUAGE MODEL

**PROBLEM:** TO PREDICT THE NEXT WORD FROM THE LAST TWO WORDS

A **Markov Model** is a table which matches possible future states of affairs with the current state of affairs of some system.

This general idea has been used for weather prediction (what are the odds it will rain tomorrow if it is raining today?) and speech and handwriting recognition.

In any language, the last two words of a piece of text can be used to predict the next, third word.

A table which gives the odds of a certain word appearing after two other words is called a **trigram model**.

# EXAMPLE: A MARKOV CHAIN LANGUAGE MODEL

## BUILDING AND TESTING A LANGUAGE MODEL

Using a Python Dictionary, we can construct a trigram model by reading a large text file containing, say English prose, and recording the most common words that come after any pair of other words. We can then use the trigram model to generate a sequence of words that sounds like English prose, sort of. It often sounds very funny.

You can read more about it in Section 3.6.4

# IN-CLASS ASSIGNMENT

## ARRAY AND SET IN PYTHON

Investigate two data types in Python: `array` and `set` in Python.

Note that in order to use `arrays` one needs to import the library, while `set` is a built-in type.

Work with them!

Try to create arrays and sets of integers. See what happens with duplicate values. Try to add/delete/replace elements.

Note that it is important to get information from *Python documentation*, not from *stack overflow*.



## HOMEWORK ASSIGNMENT, P. 104 / 9

## A SIMPLE SOLITAIRE GAME

Write a program to play the following simple solitaire game.  $N$  cards are dealt face up onto the table. If two cards have a matching rank, new cards are dealt face up on top of them. Dealing continues until the deck is empty or no two stacks have matching ranks. The player wins if all the cards are dealt. Run simulations to find the probability of winning with various values of  $N$ .