

# Lecture 22

## Chapter 12 More Python Containers

12.1 Two Familiar Containers: `list` and `tuple`

12.2 Dictionaries

12.3 Containers of containers

12.5 Arrays

# Introduction

Certain classes are designed to provide storage and management of a large collection of objects. We call them containers.

By design, Python's containers all support certain common syntaxes, yet they are different in their menu of behaviors and in the underlying efficiencies of their operations.

Let's discuss the following aspects of a container:

**order**

**mutability**

**associativity**

**heterogeneity**

**storage**

# Introduction

## **order:**

`list`, `tuple`, and `array` are used to represent *ordered sequence* of elements.

(each of these use concept of an integer *index*)

## **mutability:**

`list` is mutable (tuple is not) - we can insert, replace, or remove elements.

## **associativity:**

sometimes we need to associate an element with some data : `dict` class (a.k.a., “dictionary”) can be used for representing an association between a **key** and its **underlying value**.

## **heterogeneity:**

when elements of a container are of different types;

most Python's containers support heterogeneity

(**homogeneous** - when all elements are of the same type)

# Introduction

## storage:

for high-performance operations we don't want to have references to the values of the elements in a container (as in `List` class), but to store the actual values of elements within the state of the container;

`array` class provides compact storage for a collection of data drawn from a chosen primitive type.

	list	tuple	dict	set	frozenset	array
ordered	v	v				v
mutable	v		v	v		v
associativity			v			
heterogeneity	v	v	v	v	v	
storage						v

summary table

# 12.1 Two Familiar Containers: list and tuple

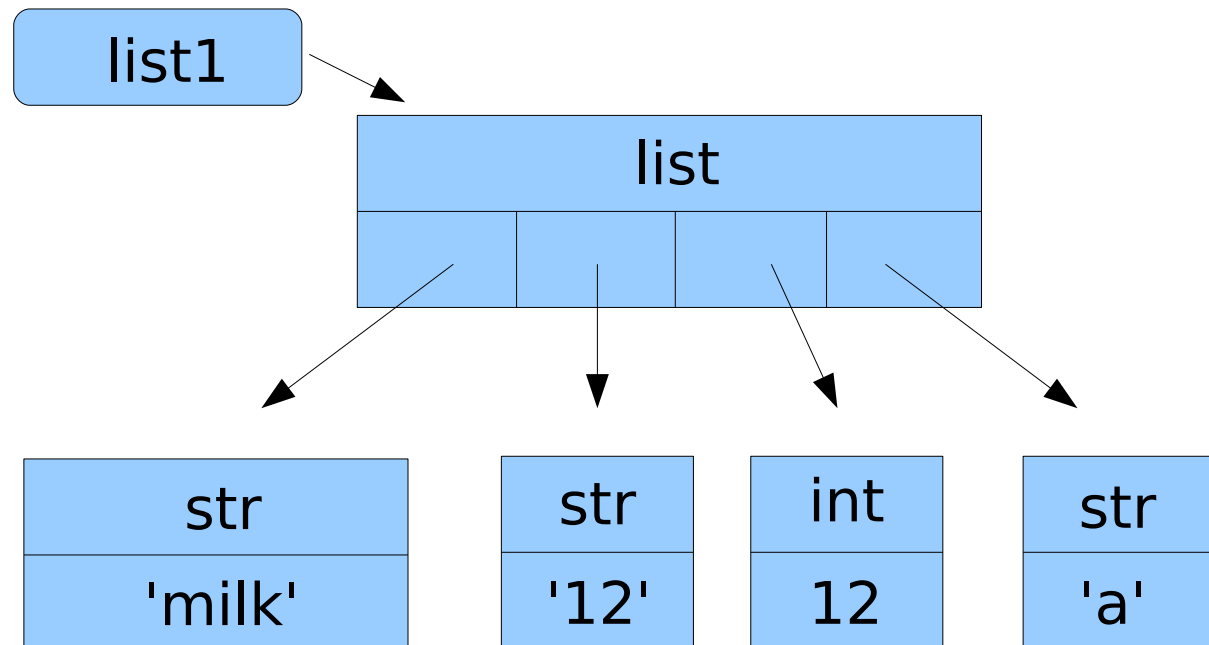
These two containers are used to manage *ordered sequence* of elements.

The position of a particular element within that sequence is designated with an *index*. Indices start with 0.

## Examples:

```
list1=['milk','12',123,'a']  
tuple1=('milk','apple','123',12)
```

```
list[1] → '12'  
tuple[3] → 12  
...
```



# 12.1 Two Familiar Containers: list and tuple

## Limitations to the use of a list and a tuple:

Assume that a company assigns each employee an “employee ID” that is an integer identifier. If the company assigns consecutive IDs (perhaps sequentially as they are hired), those IDs can be used as indices into a list.

In this way, an expression such as `employee[235]` might be used to access a particular employee's record.

**Problem 1:** when employee leaves the company  
If `employee[5]` decides to retire and is removed from the list, this causes many other employees to be repositioned within the list.

**Problem 2:** if employees' identifiers are not consecutively numbered by nature  
(some of the companies use SSN to identify employees - if they are used as indices of a list - the length of a list is 1 billion)

# 12.1 Two Familiar Containers: list and tuple

## Limitations to the use of a list and a tuple:

### **Problem 3:** non-numeric identification

If we wish to keep a track of our favorite movies and the directors of those movies.

A possible ID: title of the movie

Therefore, `director['Star Wars']` would give us the director - *but it cannot be done*

another example: list of world capitals

There is no natural numbering of countries, therefore `capital['Bolivia']` looks very natural, *but also is not supported by lists.*

## 12.2 Dictionaries

A dictionary represents a mapping from objects known as **keys** to associated objects known as **values**.

**Keys** can be consecutive integers, or can be drawn from more general domains.

**Example:** we can use dictionaries to represent mapping from movies to directors

```
director['Star Wars'] → 'George Lucas'  
director['The Godfather'] → 'Francis Ford Coppola'  
director['American Graffiti'] → 'George Lucas'
```

Title is the *key*, the name of the associated director is the *value*.

The elements of dictionaries are not inherently ordered.

# 12.2 Dictionaries

## Keys

- serve as identifiers (when accessing a particular value in collection),
- do not represent a position within the collection,
- do not need to be consecutive integers,
- do not need to be integers,
- required to be unique,
- a tuple can be used for a key
- all keys should be drawn from immutable class (int, str, or tuple)

Python uses the key when deciding where to store an entry, and also uses the key when later trying to access that entry.

## Values

No restrictions on allowable values in a dictionary

## 12.2 Dictionaries

### Syntax:

```
# create an empty dictionary, using constructor of dict class  
director = dict()
```

```
# add elements  
director['Star Wars'] = 'George Lucas'  
director['The Godfather'] = 'Francis Ford Coppola'  
director['American Graffiti'] = 'George Lucas'
```

Or

```
director = { 'Star Wars': 'George Lucas',  
            'The Godfather' : 'Francis Ford Coppola',  
            'American Graffiti' : 'George Lucas' }
```

See Table of page 404 for the list of supported behaviours

## 12.2 Dictionaries

### Examples:

```
# get the list of movie titles, sort them and display them
titles = director.keys()
titles.sort()
print titles
```

```
# print all keys elements of the dictionary 'director'
for entry in director:
    print entry
```

```
# print the sorted list of directors
for person in sorted(director.values()):
    print person, 'is a director'
```

```
# iterate over (key,value) pairs and display them
for movie, person in director.items():
    print movie, 'was directed by', person
```

## 12.3 Containers of containers

The elements of a `list`, `tuple` and the values of a `dict` class can be whatever type of object we want.

So it is perfectly acceptable to maintain a list of lists, a tuple of dictionaries, a dictionary of lists, and so on.

**Example:** Tic-tac-toe game

	0	1	2	columns
0	X	O	X	
1	X	X	O	
2	O	X	O	
rows				

`board=[[X,0,X],[X,X,0],[0,X,0]]` – row by row  
**row-major order**

or

`board=[[X,X,0],[0,X,X],[X,0,0]]` – column by column  
**column-major order**

## 12.3 Containers of containers

The elements of a `list`, `tuple` and the values of a `dict` class can be whatever type of object we want.

So it is perfectly acceptable to maintain a list of lists, a tuple of dictionaries, a dictionary of lists, and so on.

**Example:** Tic-tac-toe game

	0	1	2	columns
0	X	O	X	
1	X	X	O	
2	O	X	O	
rows				

`board=[[X,0,X],[X,X,0],[0,X,0]]` – row by row  
`board[1,2]` – 1<sup>st</sup> row, 2<sup>nd</sup> column

or

`board=[[X,X,0],[0,X,X],[X,0,0]]` – column by column  
`board[2,1]` – 2<sup>nd</sup> column, 1<sup>st</sup> row

## 12.3 Containers of containers

The elements of a `list`, `tuple` and the values of a `dict` class can be whatever type of object we want.

So it is perfectly acceptable to maintain a list of lists, a tuple of dictionaries, a dictionary of lists, and so on.

**Example:** Tic-tac-toe game

	0	1	2	columns
0	X	O	X	
1	X	X	O	
2	O	X	O	
rows				

like working  
with arrays

`board=[[X,0,X],[X,X,0],[0,X,0]]` – row by row  
`board[1,2]` – 1<sup>st</sup> row, 2<sup>nd</sup> column

or

`board=[[X,X,0],[0,X,X],[X,0,0]]` – column by column  
`board[2,1]` – 2<sup>nd</sup> column, 1<sup>st</sup> row

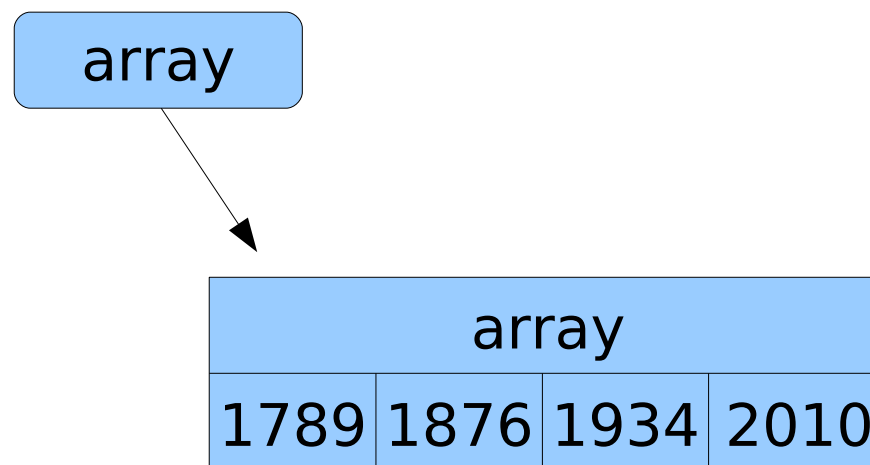
# 12.5 Arrays

The array class is not among the built-in types of Python.

```
from array import *
```

Let's create an array of integers:

```
yearArray=array('i', [1789, 1876, 1934, 2010])
```



See program [arrays-example.py](#)

# Two-dimensional data

Assume that we have the following data:

```
12  23  34  45
29  34 123  56
127 98  76 143
```

3 rows, 4 columns

integer values separated by a space

Assume that these values are stored in a file.

Let's write a program that reads all of these values and stores them somewhere.

# Two-dimensional data

```
12  23  34  45
29  34 123  56
127 98  76 143
```

Here is a sketch of the program:

```
def main():
    # open file
    fname=raw_input('input the name of the file:')
    f=open(fname)
    # store all the data in a list
    data = f.readlines()
    # form a two-dimensional array
    A=processData(data)
    # display the information
    display(A)

main()
```

# Two-dimensional data

```
12 23 34 45
29 34 123 56
127 98 76 143
```

```
12 23 34 45
```

```
29 34 123 56
```

```
127 98 76 143
```

```
data = ['12 23 34 45', '29 34 123 56', '127 98 76 143']
```

# Two-dimensional data

```
12 23 34 45
29 34 123 56
127 98 76 143
```

```
12 23 34 45
```

```
29 34 123 56
```

```
127 98 76 143
```

```
data = ['12 23 34 45', '29 34 123 56', '127 98 76 143']
```

```
12      23      34      45
```

```
29      34      123     56
```

```
127     98      76      143
```

```
data = [[12,23,34,45],[29,34,123,56],[127,98,76,143]]
```