DASC / CSE 5300

Module II

Sharma Chakravarthy
Information **T**echnology **Lab**oratory (IT Lab)
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX 76019
Email: sharmac@cse.uta.edu
URL: http://itlab.uta.edu/sharma

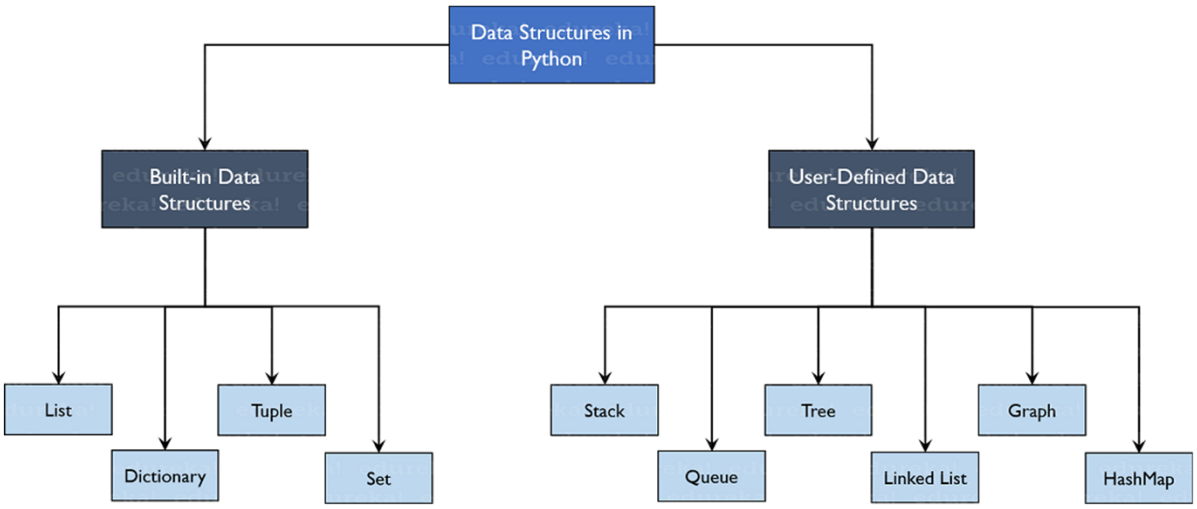
In Module II

- I will cover the following with examples so you can practice further on your own
  - Big-O complexity
  - General data structures (Linear and non-linear)
  - Python data structures for analysis
  - Algorithms for data structures
  - Basic algebra
  - Sets
  - Graphs

What is a Data Structure?

- **Organizing**, **managing** and **storing** data is important as it enables easier access and efficient modifications. Data Structures allows you to organize your data in such a way that enables you to store collections of data, relate them and perform operations on them accordingly.
- Python has **implicit** support for Data Structures which enable you to store and access data. These structures are called List, Dictionary, Tuple and Set
- Python allows its users to create their own Data Structures enabling them to have **full control** over their functionality. The most prominent Data Structures are **Stack, Queue, Tree, Linked List** and so on which are also available to you in other programming languages

Python

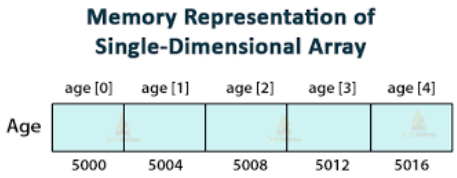

Data Structures in Python
Built-in Data Structures
User-Defined Data Structures
List
Tuple
Dictionary
Set
Stack
Tree
Graph
Queue
Linked List
HashMap

Data Structures

- From a data analysis perspective, you need to use the best and an efficient implementation of a data structure
  - Scientific data analysis
    - Arrays, multi-dimensional arrays, …
  - Tabular or transactional data analysis
    - Dataframes, tables, Lists, dictionary, …
  - Text data analysis, NLP
    - Regular expressions, strings, lists, …
  - Analysis of data with relationships preserves
    - Graphs, multilayer networks, …
- We will spend some time on each type (if not all) so you are prepared irrespective of which data type you encounter

Linear Data Structures

- Array/Vector: sequential and contiguous arrangement of homogeneous data elements along with an index for identifying the data element. Index starts from 0 in most PLs
- Length function (`len()` in Python) allows you to retrieve the length of an array
  - (avoids array bound exceeded errors)
  - Len() gives number of elements, not index which goes from 0 to len()-1
- Faster, as contiguous memory is used, but less flexible



- Widely used (sorting, stacks, list of elements, …)

Array

- Group of variables (called elements) containing values of the same type (you can put any type of Object, but have to manage it)
  - Arrays are objects, so they are reference types.
  - Elements can be either primitive or reference types.
- To access a particular element in an array
  - Use the element's index (always an integer)
  - Array-access expression—the name of the array followed by the index of the particular element in square brackets []
- Can initialize an array while declaring in Java
  - Can initialize an array of size zero in Java
  - In Python, declare and initialize at the same time

Java Array – Limitations

- The maximum size of an array has to be defined either using a constant (static or otherwise) or using a runtime variable (which is initialized)
- Once the size is defined, it cannot be changed! Remember `length` is a `final` attribute
- How do you model situations where you do not know the size of the array?
  - customers, tickets, facility rentals
  - Use ArrayList class

ArrayList class

- Java API provides several predefined data structures, called collections, used to store groups of related objects.
  - Each provides efficient methods that organize, store and retrieve your data without requiring knowledge of how the data is being stored.
  - Reduce application-development time.
- Array's do not automatically change their size (not re-sizable) at execution time to accommodate additional elements.
- ArrayList<T> (package `java.util`) can dynamically change its size to accommodate more elements.
  - `T` is a placeholder for the type of element stored in the collection.
  - This is similar to specifying the type when declaring an array, except that only non-primitive types can be used with these collection classes.
- Classes with this kind of placeholder that can be used with any type are called generic classes.
- Python also supports generic classes (built-in collection classes)

Multidimensional arrays

- Can have more than 2 dimensions
- Java does not support multidimensional arrays directly
  - Allows you to specify one-dimensional arrays whose elements are also one-dimensional arrays, thus achieving the same effect.
  - Indexing notation is different from previous languages
  - Python allows both notations

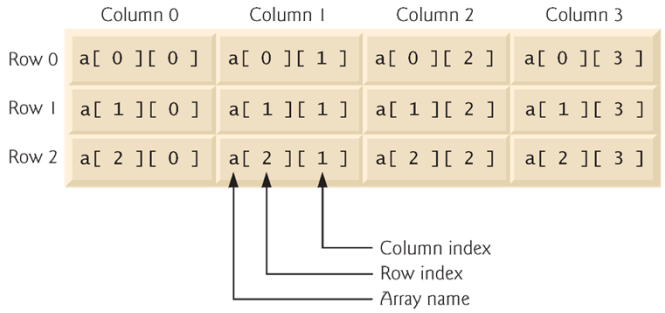


Fig. 7.16 | Two-dimensional array with three rows and four columns.

Ragged arrays

- The lengths of the rows in a two-dimensional array are not required to be the same:
  - `int[][] b = { { 1, 2, 3 }, { 4, 5 } };`
  - Each element of `b` is a reference to a one-dimensional array of `int` variables.
  - The `int` array for row `0` is a one-dimensional array with two elements (`1`, 2, and 3).
  - The `int` array for row `1` is a one-dimensional array with three elements (`4` and `5`).

1	2	3
4	5	

Vectors in Java

- Vector is **a dynamic array which can grow or shrink its size**. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework
  - Vector is synchronized
    - Only one thread can operate on a vector!
    - Increments 100% of its current size
    - Is a legacy class
  - ArrayList is non-synchronized (not thread-safe)
    - Increments 50% of its current size
    - Not a legacy class
  - Both have iterator interface

Vector operations

- Vector class in Java supports many vector operations, such as addition, subtraction, multiplication, division, dot product, cross product
- Do not get confused between vectors as used in mathematics as an object with magnitude and direction (in contrast to scalars)
- Vectors can be used for representing mathematical vectors where computation of similarity (e.g., cosine) requires dot product, and other operations

Arrays in Python

- There are several ways to initialize and use arrays in Python
  - Using [] which is essentially a list and hence can include heterogeneous elements

```
# 1st way to initialize and use an array (using range)
array =[]
array = [2 for i in range(5)]          # range with a single value generates 5 elements
array[len(array)-1] = 'strange'
array = array + ['stranger']
print(array)
print(array[2:2:1])                    # slice operator (more later)
print(array[2:3:1])
-----------------------------------------
[2, 2, 2, 2, 'strange', 'stranger']
[]
[2]
```

1. As this is a list used as an array, you Can use append to add to the array at the end.
2. You can also replace an element using index
3. You can also use .pop(index) to remove a particular element (not the same as stack.pop)

Python array/list methods

Method Name	Description
append()	Adds an item to an array
pop()	Removes an item from an array
clear()	Removes all items from an array
copy()	Returns a copy of an array
count()	Returns the number of elements in a list
index()	Returns the index of the first element with a specific value
insert()	Adds an element to the array at a specific position
reverse()	Reverses the order of the array
sort()	Sorts the list

Arrays in Python

➢ array module in Python supports arrays

```
# 2nd  way to initialize and use an  array using array module
import array as arr
 # creating an array with char type
a = arr.array('B', [65, 66, 67, 68])
 # printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 4):
    print ("{:c}".format(a[i]), end = ' ')
 # creating an array with float type
b = arr.array('d', [2.5, 3.2, 3.3, 45, 'dasc'])
b = arr.array('d', [2.5, 3.2, 3.3, 45])
b.insert(1, 100)
for i in range (0, 5):
    print (b[i], end =" ")
```

```
The new created array is :  A B C D
2.5 100.0 3.2 3.3 45.0
```

Remove operation is also allowed at an index

```
TypeError                                 Traceback (most recent call last)
<ipython-input-21-bfc2f58bf8ab> in <module>()
     12
     13 # creating an array with float type
---> 14 b = arr.array('d', [2.5, 3.2, 3.3, 45, 'dasc'])
     15 # printing original array
     16 print ("The new created array is : ", end =" ")

TypeError: must be real number, not str
```

Type codes of array module

Type code	C Type	Python Type	Minimum size in bytes	Notes
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	wchar_t	Unicode character	2	16 or 32 bits based on the platform
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	
'Q'	unsigned long long	int	8	
'f'	float	float	4	
'd'	double	float	8	

Arrays in Python

- NumPy (numerical Python) library supports arrays
  - NumPy (**Numerical Python**) is an open source Python library that is used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems
  - The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.
  - The NumPy library contains multidimensional array and matrix data structures
  - It provides **ndarray (as the data type)**, a homogeneous n-dimensional array object, with methods to efficiently operate on it
  - Faster and more compact than python lists

NumPy Arrays: basics

```
np.array()    # basic function
np.zeros(k)   # an array of k elements initialized to 0.
np.zeros((3,4))      # can be multi-dimensional
np.ones(k)    # an array of k elements initialized to 1.
Np.ones((2, 3, 4, 5))      # multi-dimensional
np.empty()    # fills array of k elements with 0
np.random.random((2,3))  # fills array of k elements with random values;
                         # faster than filling 0
np.arange(start, stop, incr)      # from start to < stop in incr steps
np.linspace(begin, end, step)  #spaced linearly in a specified interval
```

Default data type is floating point (np.float64), you can explicitly specify the data type you want using the `dtype` keyword

```
a = np.ones(2, dtype = np.int64)
np.sort(arr)     # sorts arr in ascending order
np.concatenate(arr1, arr2)#concatenates aar2 to arr1
```

Array Examples in Python

```
a = np.array([1, 2, 3, 4, 5, 6])          # only one axis
b = np.array([ [1, 2, 3,4],               # b[0]
               [5, 6, 7, 8],              # b[1]
               [9, 10, 11, 12] ]          # b[2]
Print(b[0])                               # prints axis 1 of length 4
d = arange(4, 49, 5)          # 4 to < 49 in steps of 5
Print(d)
e = np.linspace(1, 123, 11)  # 11 linearly spaced values in 1 to 123
Print(e)
c = np.full((3,5,4),7)
Print(c)
```

Shape of b is (3, 5, 4)

```
[1, 2, 3, 4]
[ 4 9 14 19 24 29 34 39 44]
[ 1. 13.2 25.4 37.6 49.8 62. 74.2 86.4 98.6 110.8 123. ]
```

```
[[[7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]]
 [[7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]]
 [[7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]
  [7 7 7 7]]]
```

Not A Number (NaN)

- This is different from None
- A special value in NumPy that represents values that are not numbers.
- It is called NaN
- A place holder for empty spaces
- Can represent a missing value at that place
  - Very useful for data analysis
- Check for Nan (.isnan()) is available in Pandas, NumPy and math packages
  - Math load is smaller than the other two
- Useful for representing missing values as NaN (instead of 0) and filter them out in pre-processing

Not A Number (NaN) Usage

```
d = np.zeros(10)    # initializes elements of an array t
0
print(d)
d[1] = math.nan     # changes to Not a Number
print(d[1])
print(math.isnan(d[1]))
d[1] = 100
print(math.isnan(d[1]))
d[2] = None
print(math.isnan(d[2]))
d = np.random.random(5)
print(d)
-----------------------------------
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
nan
True
False
True
[0.50612339 0.02509018 0.7930157  0.9166238  0.12907751]
```

What is filled for d = np.empty(5)?

What happens for d[3] = 'dasc'?

`numpy.all(numpy.isnan(data_list))`
checks if all elements in the list are nan

Arithmetic Operations

- +, -, *, and / are supported
- Can combine scalar and matrix for the above. Scalar will be applied for every element of the array

```
Import numpy as np
a = np.array([  [10, 20, 30],
                [40, 50, 60]  ]
print(a + 2)
print(a – 10)
print(a*2)
print(a/20)
```

```
[[12 22 32]
 [42 52 62]]
[ [0 10 20]
  [30 40 50]]
[[20 40 60]
 [80 100 120]]
[[0.5 1. 1.5]
 [2. 2.5 3.]]
```

Arithmetic Operations

- For matrix operations, shapes need to be taken into consideration. They don't have to be the same, but there has to be a reasonable way of performing these operations

```
Import numpy as np
a = np.array([  [10, 20, 30],
                [40, 50, 60]  ]
b = np.array([  [1, 2, 3]
c= np.array([  [1],
               [2]  ]
print(a + b)
print(a + c)
print(a*b)
Print(a/b)
print(a/c)
```

```
[[11 22 33]
 [41 52 63]]
[[11 21 31]
 [42 52 62]]
[[ 10  40  90]
 [ 40 100 180]]
[[10. 10. 10.]
 [40. 25. 20.]]
[[10. 20. 30.]
 [20. 25. 30.]]
```

There is also numpy.matrix
Can get transpose, multiplicative inverse, and conjugate transpose of self

Attributes of NumPy Arrays

`b.ndim` # number of dimensions of b

returns 3 for the previous example

`ndarray.size`

will return the total number of elements of the array. This is the *product* of the elements of the array's shape

`ndarray.shape`

will display a tuple of integers that indicate the number of elements stored along each dimension of the array. If, for example, you have a 2-D array with 2 rows and 3 columns, the shape of your array is (2, 3).

```
ndarray.dtype # returns the data type of values in the
                  array
```

Shape Manipulation

- Allows to convert a 1 axis (dimension array) into a multi-dimensional array

a = [0 1 2 3 4 5]

b = a.reshape(3,2) will convert it into 3 rows and 2 columns

b = [[0 1]
[2 3]
[4 5]]

```
Numpy.reshape(a, newshape =(1,6), order = 'C')
```

a is the array to be reshaped.

newshape is the new shape you want. You can specify an integer or a tuple of integers. If you specify an integer, the result will be an array of that length. The shape should be compatible with the original shape

order: C means to read/write the elements using C-like index order, F means to read/write the elements using Fortran-like index order, A means to read/write the elements in Fortran-like index order if a is Fortran contiguous in memory, C-like order otherwise. (This is an optional parameter and doesn't need to be specified.)

Mathematical Functions

`np.exp(a)` takes e to the power of each value

`np.sin(a)`
`np.cos(a)`
`np.tan(a)`
`np.log(a)`
`np.sqrt(a)`

`a.sum()`
`a.min()`
`a.max()`
`a.mean()`
`np.median(a)`
`np.std(a)`

Note calls with parameters and calls using module name. They seem to be interchangeable

`Print(a.median())` gives error

`np.sum(a))` `a.sum()` gives the same value!

Other operators

- You can choose elements based an on expression

Print(a[a > 12*2]) will print all elements in the array that is greater than 24

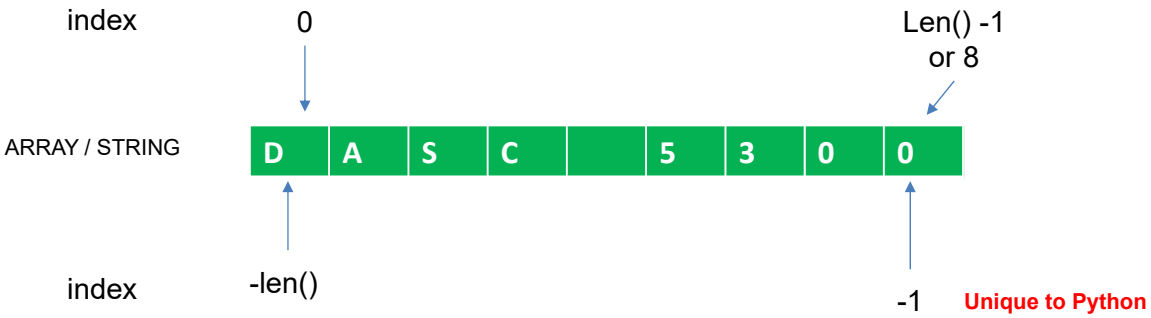
- You can stack arrays vertically (vstack) or horizontally (hstack)
- You can split arrays vertically and horizontally
- You can do scalar, vector operations; multiplying an array with a scalar value
- You can compute maximum (.max), minimum (.min), sum, mean, product, and std
- You can get unique values, flip, and flatten arrays

Indexing and slicing

- This is the same for Python lists
- In Python array, there are multiple ways to print the whole array with all the elements, but to print a specific range of elements from the array, we use Slice operation
- Slice operation is performed on array with the use of colon(:)
- To print elements from beginning to a range use [:Index],
- to print elements from end use [:-Index]
- to print elements from specific Index till the end use [Index:]
- to print elements within a range, use [Start Index:End Index], and
- to print whole List with the use of slicing operation, use [:]. Further, to print whole array in reverse order, use [::-1].

Python: indexing

- Python index starts from 0 as most other languages
  - Same for arrays, strings, lists, … Not sets (Why?)
  - Goes from 0 to len(a) -1; also from -1 to –len()



  - Gives ArrayBoundException error in other languages

Indexing and slicing

- This is the same for Python lists

Syntax: [start : stop : steps]

which means that slicing will start from index start
will go up to stop in step of steps.
Default value of start is 0, stop is last index of list
and for step it is 1

[:stop] will slice from 0 up to stop

[start:] will slice from start index till end

Negative value of steps shows right to left traversal instead of left to right

[::-1] will print the list in reverse order

can also be used for checking palindrome! (string[:] == string[::-1])

Start value is -1, stop value is –len(), step is -1

For any list/string a: a[::-1] will reverse it (could have been used for palindrome check if , and space were not there!)

Indexing and slicing

- NumPy arrays when sliced return views (rather than copies of data)
  - If the slice is modified, original array is modified
  - However, slicing of Python lists always returns a copy
    - Modification of the slice does not affect the original
- When you are dealing with large data sets, you do not want copies (memory requirement goes up)
- Views are good as you can take partitions and play with them including modification

Loading and Saving Arrays

- You can save a serialized form of an arry on disk to load it later for use
  - We discussed serialization in Python earlier

Import numpy as np

```
a = np.array([
                [10, 20, 20]
                [40, 50, 60]
                [70, 80, 90]
                [100, 110, 120]])
np.save('myarray.bin', a) # extension can be anything
```

Load it at a later time using

```
np.load('myarray.bin')
```

Save an object in CSV format using

```
np.savetxt('myarray.csv', a)
```

Load using

```
np.loadtxt('myarray.csv')
```

Questions/comments

For more information visit:

http://itlab.uta.edu