Data Structures-1

Prof. K. Adisesha

BE, M.Sc., M.Th., NET, (Ph.D.)

Learning objectives

- **Introduction**
- **Data Structure Types**
  - **Primitive Data Structure**
  - **Non-Primitive Data Structure**
- **List Manipulations**
- **Data Structure Operations**

Data Structure

What is Data Structure ?

- Data structures are a way of organizing and storing data so that they can be accessed and worked with efficiently.
- Data structures and algorithms are closely related
  - Representation and organization of data
  - Facilitate access and modification of data
  - Different data structures have strengths and weaknesses
  - Better suited for a specific algorithm than others

Data Structure

Data Structure using Python

- We will use Python as programming language
- Data structures and algorithms are independent of programming language
  - This is not a basic programming course
  - We focus on higher level issues
  - You should already have experience with a programming language (Python, Java, C/C++)

Data Structure

Primitive Data Structures

- These are the most primitive or the basic data structures.
- They are the building blocks for data manipulation and contain pure, simple values of a data.
- Python has four primitive variable types:
  - Integers
  - Float
  - Strings
  - Boolean

Data Structure

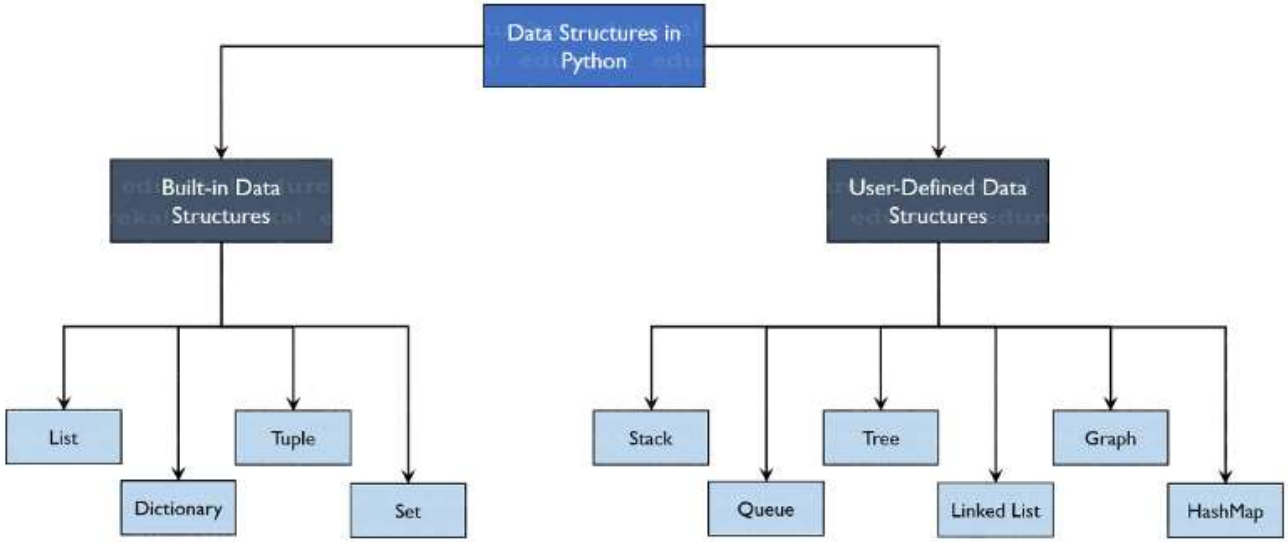
Non-Primitive Data Structures

- Non-Primitive Data Structures are the sophisticated members of the data structure family.
- They don't just store a value, but rather a collection of values in various formats.
- In the traditional computer science world, the non-primitive data structures are divided into:
  - Arrays
  - Lists
  - Tuples
  - Dictionary
  - Sets
  - Files

Data Structure

Types of Data Structures in Python

- Built-in Data Structures
- User-Defined Data Structures



Data Structure

Built-in Data Structures

- Data Structures are built-in with Python which makes programming easier and helps programmers use them to obtain solutions faster.
- Built-in Data Structures in Python
  - List
  - Dictionary
  - Tuple
  - Sets

List

LIST

- Lists in Python are used to store collection of heterogeneous items.
- These are mutable, which means that you can change their content without changing their identity.
- You can recognize lists by their square brackets [and] that hold elements, separated by a comma ,.
- There are addresses assigned to every element of the list, which is called as Index.
- The index value starts from 0 and goes on until the last element called the positive index.
- There is also negative indexing which starts from -1 enabling you to access elements from the last to first.

List Manipulation

List Manipulation

- Python provides many methods to manipulate and work with lists.
- Common list manipulations Functions are:
  - Adding new items to a list,
  - Removing some items from a list,
  - Sorting or reversing a list are.
  - Find length of the list.
  - Find the index value of value passed in List.
  - Find the count of the value passed to List.

List Manipulation

Adding Elements

- Adding the elements in the list can be achieved using:
  - **append() function**
  - **extend() function**
  - **insert() function**

List Manipulation

append() function

- The append() function adds all the elements passed to it as a single element.
- Example:

```
my_list = [1, 2, 3]
print(my_list)
my_list.append([4, 5,6]) #add as a single element
print(my_list)
```

  - **Output:**

```
[1, 2, 3]
[1, 2, 3, [4, 5, 6]]
```

List Manipulation

extend() function

- The extend() function adds the elements one-by-one into the list.

- Example:

```
my_list = [1, 2, 3]
print(my_list)
my_list.extend([4, 5])  #add as different elements
print(my_list)
```

  - **Output:**

```
[1, 2, 3]
[1, 2, 3, 4, 5]
```

List Manipulation

insert() function

- The insert() function adds the element passed to the index value and increase the size of the list too.

- Example:

```
my_list = [1, 2, 3]
print(my_list)
my_list.insert(1, 'Sunny') #add element at 1
print(my_list)
```

  - **Output:**

```
[1, 2, 3]
[1, 'Sunny', 2, 3, 4, 5]
```

List Manipulation

Deleting Elements

- **del keyword**
  - To delete elements, use the **del keyword** which is built-in into Python but this does not return anything back to us.
- **pop() function**
  - If you want the element back, you use the **pop() function** which takes the index value.
- **remove() function.**
  - To remove an element by its value, you use the **remove() function**.
- **clear() function**
  - To remove all elements from the list, to make an empty list we use the **clear() function**.

List Manipulation

del keyword

- To delete elements, use the **del keyword** which is built-in into Python but this does not return anything back to us.
- Example:

```
my_list = [10, 'Sunny', 20, 30, 40, 50]
print(my_list)
del my_list[5] #delete element at index 4
print(my_list)
```

  - **Output:**

```
[10, 'Sunny', 20, 30, 40, 50]
[10, 'Sunny', 20, 30, 50]  #after deleting index 4
```

List Manipulation

pop() function

- If you want the element back, you use the **pop() function** which takes the index value.
- Example:

```
my_list = [10, 'Sunny', 20, 30, 40, 50]
print(my_list)
a = my_list.pop(1) #pop element from list
print('Popped Element: ', a, ' List remaining: ', my_list)
```

  - **Output:**

```
[10, 'Sunny', 20, 30, 40, 50]
'Popped Element: Sunny List remaining: [10, 20, 30, 40, 50]
```

List Manipulation

remove() function

- To remove an element by its value, you use the **remove() function**.
- Example:

```
my_list = [10, 'Sunny', 20, 30, 40, 50]
print(my_list)
my_list.remove('Sunny') #remove element with value
print(my_list)
```

  - **Output:**

```
[10, 'Sunny', 20, 30, 40, 50]
[10, 20, 30, 40, 50]        #after deleting 'Sunny'
```

List Manipulation

clear() function

- To remove all elements from the list, to make an empty list we use the **clear() function**.
- Example:

```
my_list = [10, 'Sunny', 20, 30, 40, 50]
print(my_list)
my_list.clear() #empty the list
print(my_list)
```

  - **Output:**

```
[10, 'Sunny', 20, 30, 40, 50]
[ ]          #empty the list
```

List Manipulation

Accessing Elements

- Accessing elements from a List is done by range of indexes by specifying start and end position of the range.
- When specifying a range, the return value will be a new list with the specified items.
- Example:

```
my_list = [10, 20, 30, 40, 50]
for element in my_list:        #access elements one by one
    print(element)
```

Output:

```
10, 20, 30, 40, 50
```

List Manipulation

Accessing Elements

- You pass the index values and hence can obtain the values as needed.
- Example:

```
my_list = [10, ['Sunny', 20, 30, 40], 50]
print(my_list)
```

Output: [10, ['Sunny', 20, 30, 40], 50]

```
len(my_list)
```

Output: 3

```
print(my_list[1])
```

Output: ['Sunny', 20, 30, 40]

```
print([1][0])
```

Output: ['Sunny']

```
print([1][0][-4])
```

Output: 'u'

List Manipulation

Copy a List:

- There are ways to make a copy, one-way is to use the built-in copy(), list() method.
- **copy() method**: Make a copy of a list:
  - **Example**

```
StuList = ["Prajwal", "Sunny", "Rekha"]
mylist = StuList.copy()
print(mylist)
```

Output: ['Prajwal', 'Sunny', 'Rekha']

- **list() method:** Make a copy of a list:
  - **Example**

```
StuList = ["Prajwal", "Sunny", "Rekha"]
mylist = list(StuList)
print(mylist)
```

Output: ['Prajwal', 'Sunny', 'Rekha']

List Manipulation

Count(): You can count the number of element of a kind**:**

- **Example**

```
my_list = [10, 20, 10, 40, 10]
my_list.count(10)
```

Output: 3

Sort(): There is a sort() method that performs an in-place sorting:

- **Example**

```
StuList = my_list = [10, 20, 10, 40, 10]
my_list.sort()
print(my_list)
```

Output: [10, 10, 10, 20, 40]

Reverse: Finally, you can reverse the element in-place:

- **Example**

```
my_list = ['a', 'c' ,'b']
my_list.reverse()
print(my_list)
```

Output: ['b', 'c', 'a']

List Methods

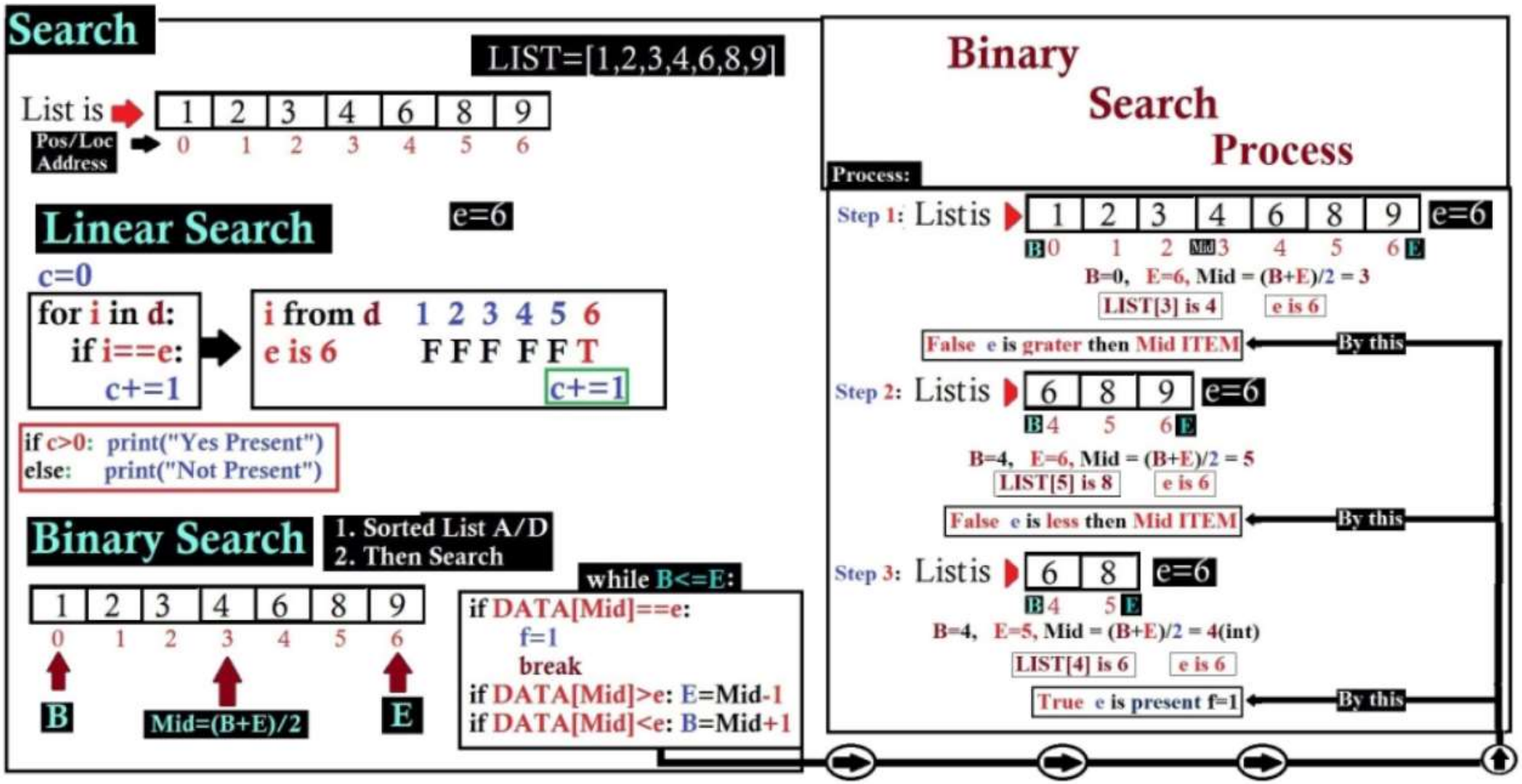
Built-in Python list methods

- Following is the table containing the set of built-in methods that you can use on List.

Method	Description
append()	It adds a new element to the end of the list.
extend()	It extends a list by adding elements from another list.
insert()	It injects a new element at the desired index.
remove()	It deletes the desired element from the list.
pop()	It removes as well as returns an item from the given position.
clear()	It flushes out all elements of a list.
count()	It returns the total no. of elements passed as an argument.
sort()	It orders the elements of a list in an ascending manner.
reverse()	It inverts the order of the elements in a list.
copy()	It performs a shallow copy of the list and returns.
len()	The return value is the size of the list.

SEARCHING

SEARCHING ALGORITHMS



SORTING

SORTING ALGORITHMS

Bubble Sort

```
x =[3,56,2,156,78,56,34,23,41,78,8,123,45]
for i in range(1,len(x)):
    for j in range(0,len(x)-i):
        if x[j] > x[j+1]:
```

Ascending >
Descending <

Method 1:
```
x[j],x[j+1] = x[j+1],x[j]
```

Method 2:
```
t=x[j+1]
x[j+1]=x[j]
x[j]=t
```

same

Selection Sort

```
for i in range(0,len(x)-1):
    pos = i
    low = x[pos]
    for j in range(i+1,len(x)):
        if low > x[j]:
            pos = j
            low = x[j]
x[i],x[pos] = x[pos],x[i]
```

Merge Sort

```
x = [1,3,4,12,15,22]
y = [2,5,6,8,10,19]
z = []
i=j=k=p=0
```

```
while 1:
    if x[i]<=y[j]:
        z.append(x[i])
        i = i+1
```

```
else:
    z.append(y[j])
    j = j+1
p+=1
if p>=(len(x)+len(x))-1:
    break
```

Conclusion!

- We learned about:
  - Data Structure Definition
  - Data Structure Types
  - Primitive Data Structure
  - Non-Primitive Data Structure
  - List Manipulations
  - Data Structure Operations

Thank you