uc3m | Universidad Carlos III de Madrid

Unit 2
Singly Linked List

Data Structures and Algorithms

Linear ADTs

- Stack ADT
- Queue ADT
- **Singly Linked List ADT**
- Doubly Linked List ADT

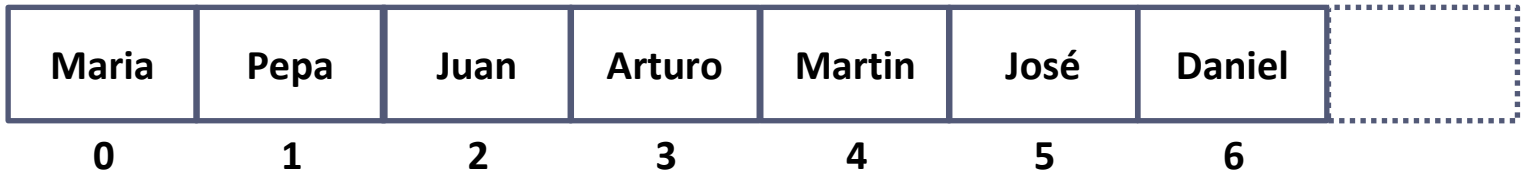
Agenda

- **The common problems with Python lists implementation**
- Definition of List ADT
- Implementing a singly linked list ADT

The common problems with Python lists implementation

Python Lists are saved as **consecutive cells** in memory (consecutive way, **one after the other with no gap** (without interruption), the first item has index [0], the second item has index [1] etc..

A = ["Maria", "Pepa", "Juan", "Arturo", "Martin", "Jose", "Daniel"]



Easy and fast access to all its elements:

print(A[3]) => Arturo

print(A[6]) => Daniel

The common problems with Python lists implementation

1st Problem:

- **pop** needs to move all the elements one spot to the left in the list.
- What happens, if the list is very large, or your program has to perform a lot of 'remove' operations? **pop is not efficient!**

pop(0)

The common problems with Python lists implementation

2nd Problem:

- **insert** needs to move all the elements one spot to the right
- What happens, if the list is very large, or your program has to perform a lot of 'insert' operations? **insert is not efficient!**

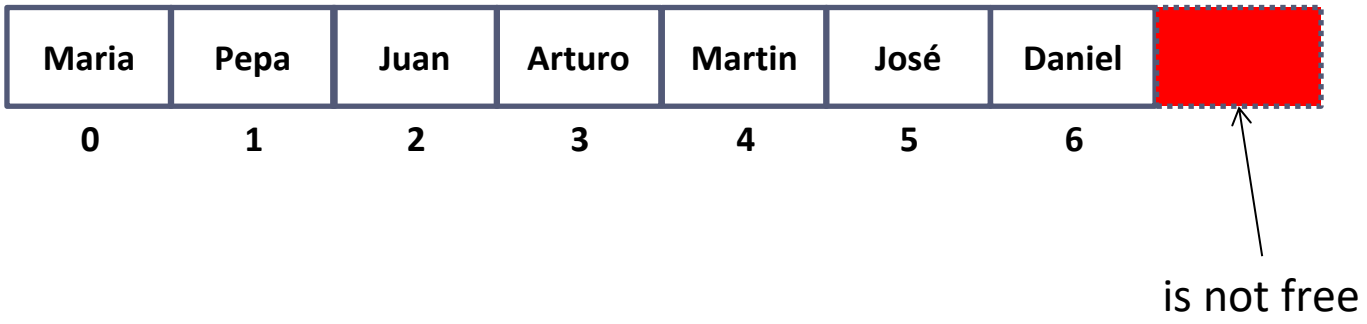
insert(3,"Isabel")

The common problems with Python lists implementation

3rd Problem:

- If the next location after the last element is not free.
- Python has to find enough space to copy all the elements of the list and then adds the new element. **it is not efficient!**

append("Isabel")



Agenda

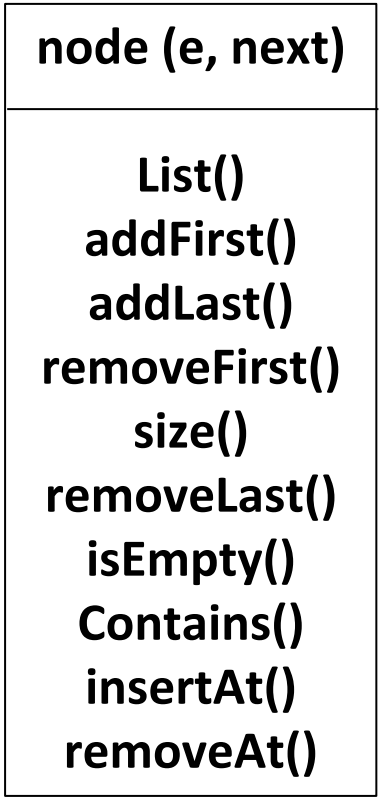
- The common problems with Python lists implementation
- **Definition of List ADT**
- Implementing a singly linked list ADT

Definition of List ADT

- a collection of nodes linked together in **a sequential way**, and each node has two parts; one part is a data part, and another part is an address.

Some possible operation are:

- **List**() creates a new list.
- **addFirst**(L,e) add the element e at the beginning of the list L.
- **addLast**(L,e) add the element e at the tail of the list L.
- **removeFirst**(L) removes the first element of the list L. It returns the element.
- **size**(L): returns the number of items of the list.



List ADT

Definition of List ADT

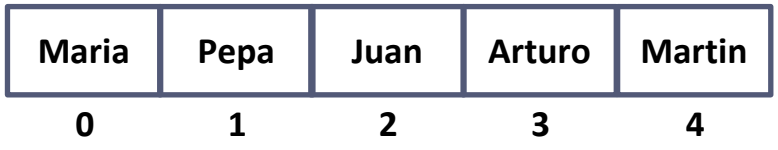
More operations:

- **removeLast**(L) removes the last element of the list L. It returns the element.
- **isEmpty**(L): returns True if the list L is empty, False otherwise.
- **contains**(L,e): returns the first position of the element e in the list. If the element doesn't exist return -1.
- **insertAt**(L,index,e): inserts the element e at the position index of the list L.
- **removeAt**(L,index): removes the element at the position index of the list L. It returns the element.

Agenda

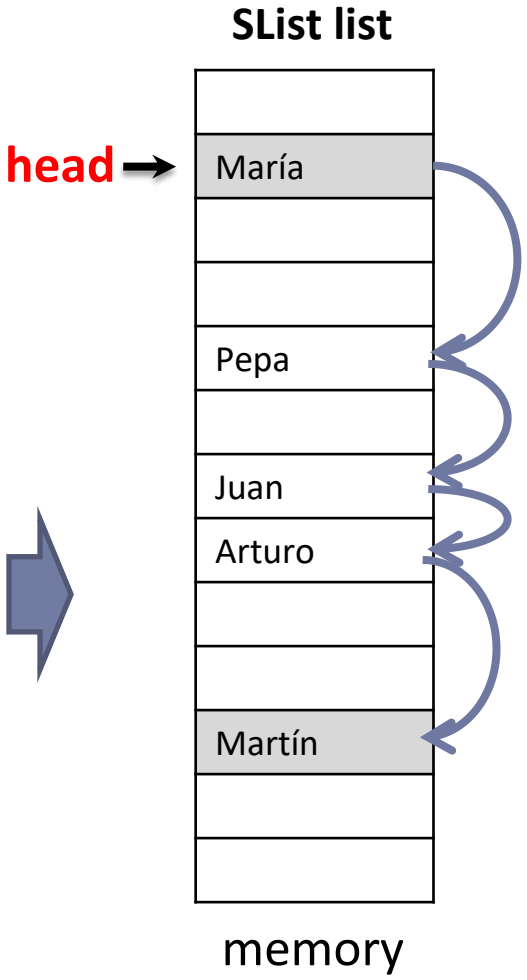
- The common problems with Python lists implementation
- Definition of List ADT
- **Implementing a singly linked list ADT**

Implementing a Singly Linked List ADT



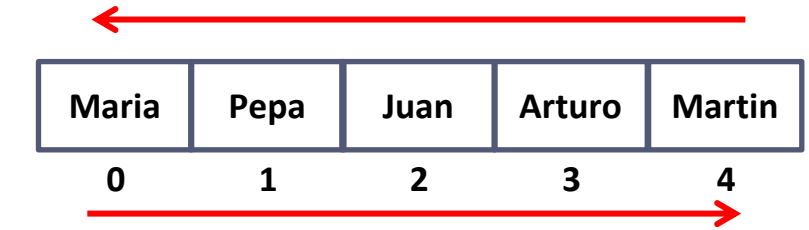
Python list

one after the other with no gap



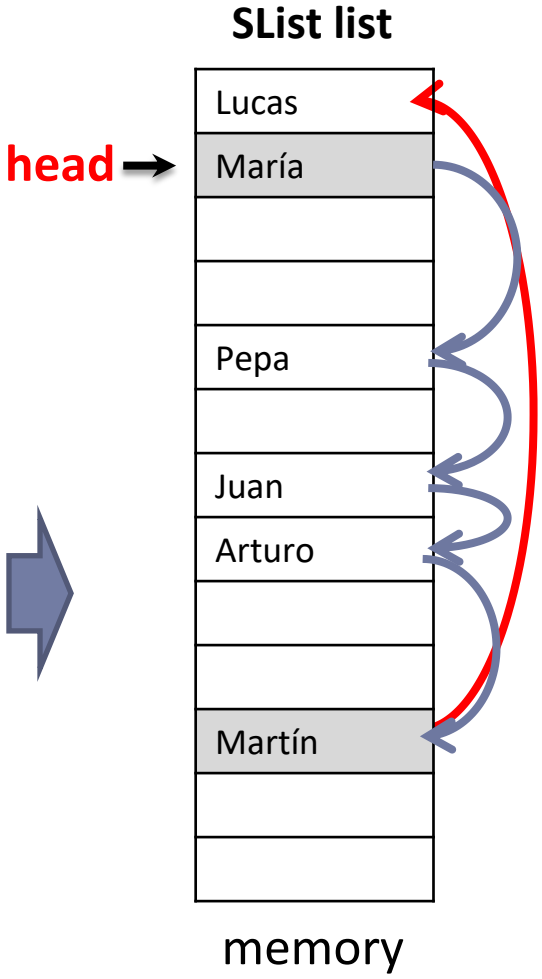
- It allows gaps between the different nodes of the list. **No consecutive way**
- 'head' is used to access the first node of the list. **NO index**.
- To connect between the nodes of the list, **references** are used.

Implementing a Singly Linked List ADT



Python list

add/remove, move the nodes to the right or to the left

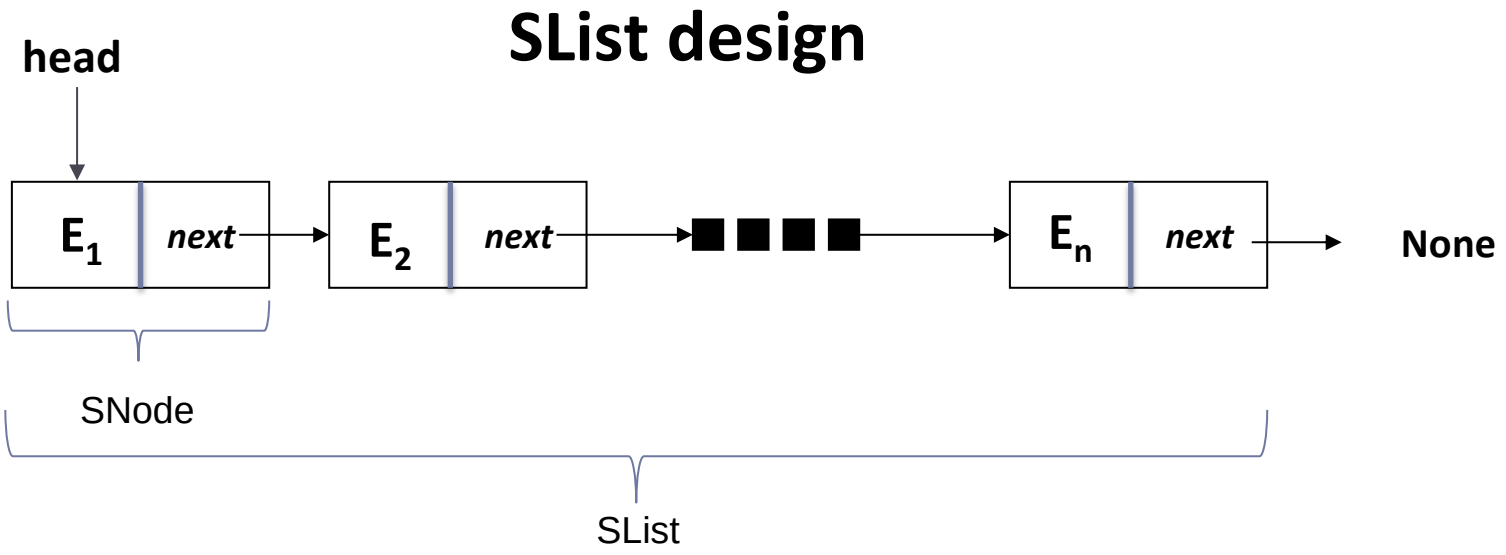


addLast("Lucas")

- There is no need to move the nodes to the right or to the left.
- Python finds a new location for the new node, and then save it in this location
- The last node of the list points to the new node.

Implementing a Singly Linked List ADT

- Each node stores an element of the sequence and a reference to the next node of the list.
- The list uses a reference to the first node, named **head**.



Implementing a Singly Linked List ADT

```
class SList:
  def __init__(self):
    self.head=None
    self.size=0
```

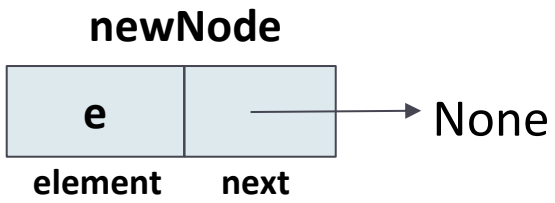
```
class SNode:
  def __init__(self, e, next=None):
    self.elem = e
    self.next = next
```

```
l=SList()
```

```
l.size=0
```

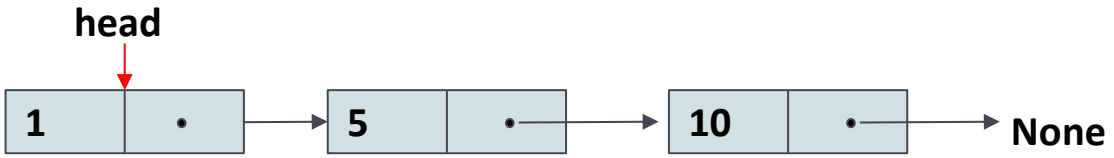
l.head → None

```
newNode=SNode(e)
```



addFirst(L,e)

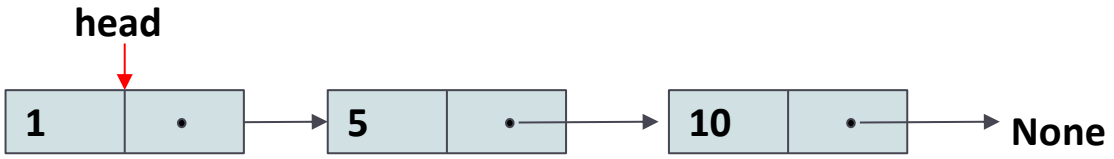
```
def addFirst(self,e):
        newNode=SNode(e)
        newNode.next=self.head
        self.head=newNode
        self.size=self.size+1
```



addFirst(L,e)

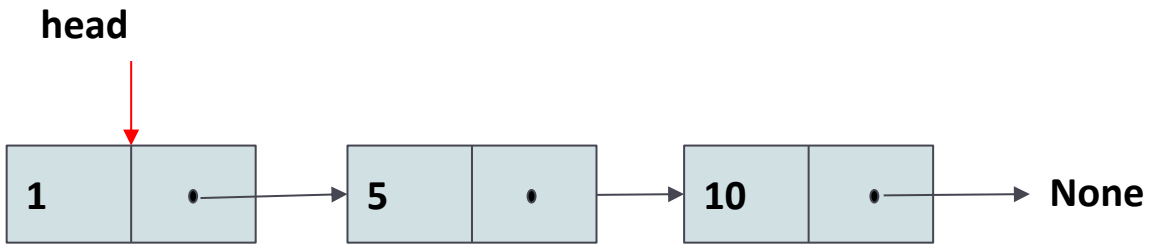
```
def addFirst(self,e):
        newNode=SNode(e)
        self.head=newNode
        newNode.next=self.head
        self.size=self.size+1
```

Does the order matter?



removeFirst(L)

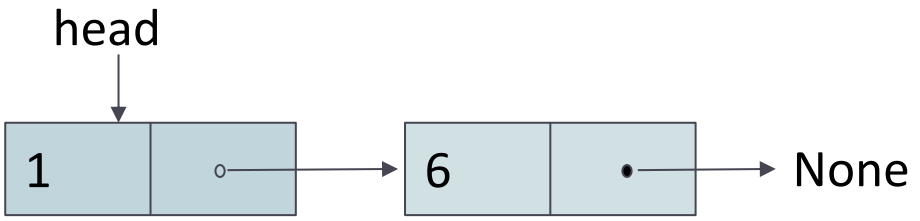
```
def removeFirst(self):
   result = None
   if self.isEmpty():
     print("List is emtpy!!!")
   else:
     result = self.head.elem
     self.head = self.head.next
     self.size -= 1
   return result
```



addLast(L,e)

```
def addLast(self,e):
  if self.isEmpty():
    self.addFirst(e)
  else:
    newNode=SNode(e)
    node=self.head
    while node.next != None:
            node=node.next

    node.next=newNode
    self.size +=1
```

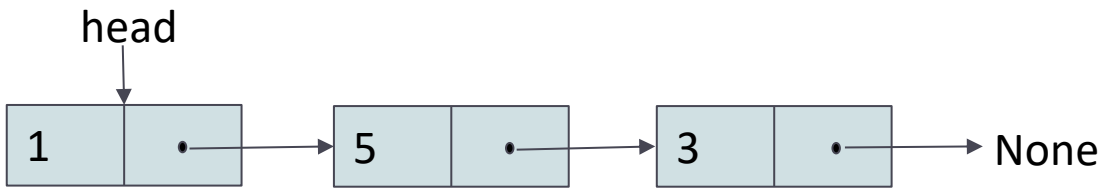


The last node is reached when node.next=None

removeLast(L)

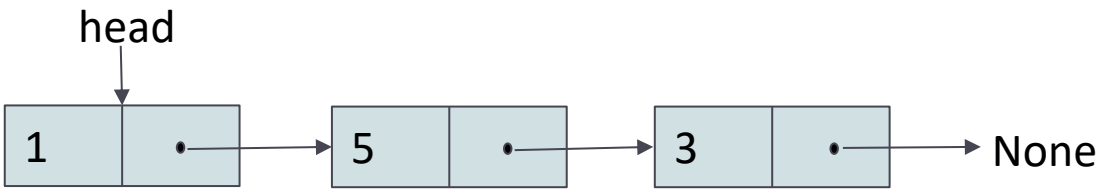
```
def removeLast(self):
  result=None
  if self.isEmpty():
    print("List is emtpy!!!")
  else:
    penult=None
    last=self.head
    while last.next != None:
      penult=last
      last=last.next

    if penult==None:
      # The list only has an element
      result=self.removeFirst()
    else:
      result=last.elem
      penult.next=None
      self.size -=1
  return result
```



getAt(L,index)

```
def getAt(self,index):
  if index<0 or index>=self.size:
     print(index,'error: index out of range')
     return None
  i=0
  node=self.head
  while  i<index:
     node=node.next
     i+=1
return node.elem
```



Exercises for the lab class

- Implement the following methods:
  - contains(L,e):returns the first position of the element e at the list. If e does not exist, then it returns -1.
  - insertAt(L,index,e): inserts the element e at the position index of the list L.
  - removeAt(L,index): removes the element at the position index from the list L.

Exercises for the lab class

- Implement a stack with a singly linked list.
- Implement a queue with a singly linked list.