Python Lists & Loops

Lists

- List items are ordered, changeable, and allow duplicate values.

```
fruits = ["apple", "banana", “cherry"]
```

- List items are indexed
  - the first item has index `[0]`, the second item has index `[1]` etc.

```
>>> fruits[2]
‘cherry’
```

- Lists are mutable

```
>>> fruits[1]='plum'
>>> fruits
['apple', 'plum', ‘cherry']
>>> len(fruits)
3
```

Lists

append(), pop() & extend()

```
>>> fruits = ["apple","banana","cherry"]
>>> fruits
['apple', 'banana', ‘cherry']
>>> fruits.append('mango')
>>> fruits
['apple', 'banana', 'cherry', 'mango']
>>> x=fruits.pop()
>>> x
'mango'
>>> fruits
['apple', 'banana', 'cherry']
>>> fruits2=['mango','pear']
>>> fruits.extend(fruits2)
>>> fruits
['apple', 'banana', 'cherry', 'mango', 'pear']
```

Accessing List Elements & Slicing

- fruits[-1] - last element (-i mean len-i)
- fruits[i:j]   new sublist called a slice consisting of items from i to j-i
- eg if fruits=['apple', 'plum', 'cherry', 'mango', ‘pear']
- fruits[1:3] is ['plum', 'cherry']
- fruits[1:] means slice till end of the list similar fruits[:2] from beginning
- `concatenation +`
- `fruits[1:] + fruits[:2]`

```
['plum', 'cherry', 'mango', 'pear', 'apple', 'plum']
```

List Data Types

- A list can contain different data types:

```
>>>list = ["abc", 34, True, 40, “male”]

>>>type(list)

<class ‘list'>

>>> type(list[0])

<class ‘str'>

>>> type(list[1])

<class 'int'>
```

Complexity of Basic List Operations

Get Item O(1)
Set Item O(1)
Get Length O(1)

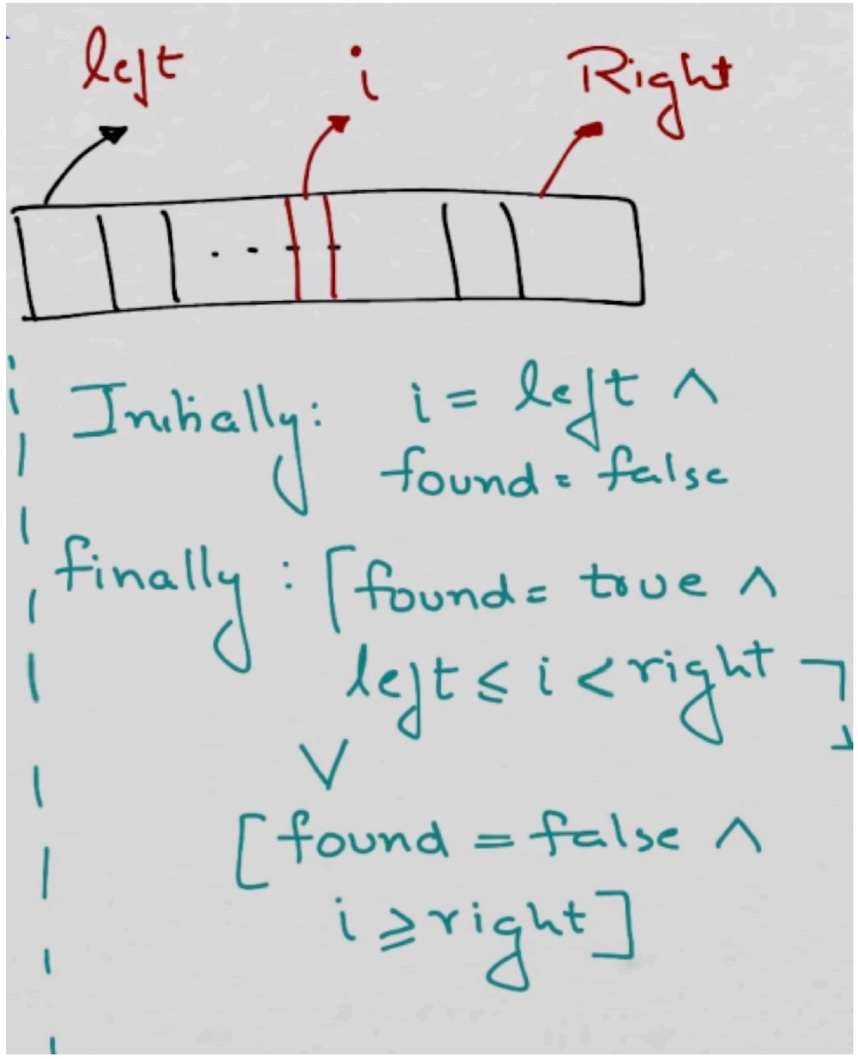
Append* O(1) [*on average]
Pop last O(1)

Extend O(k)
Get Slice O(k)

Sequential Search

Search for x in List A between left & right position

```
def SeqSearch (A, x, left, right):
```



Sequential Search

Search for x in List A between left & right position

```
def SeqSearch (A, x, left, right):
    1# assert: A[left,...,right-1] and
               x is established
    i = left
    found = False
    3# found = false ⇔
       (x ∉ A[left, ..., i-1] ∧ left ≤ i ≤ right)
    while ((not found) and
           (i < right)):
        if (A[i] == x):
            found = True
        Else:
            i = i+1
    2# assert: (found at i) or (x ∉ A[left,...
                                    right])

if found:
    print i
Else:
    print "not found"
```

Initially: $i = left \wedge found = false$

finally: $[found = true \wedge left \leq i < right] \vee [found = false \wedge i \geq right]$

For loop and range

- **When you want to do something for every item of a list you can use the for statement**

```
for item in L:
        S
def seqSearch(A,x,left,right):
    found=false
    for item in A[left:right]:
        if item == x:
            found= true
            break
    return found
```

range

- The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number

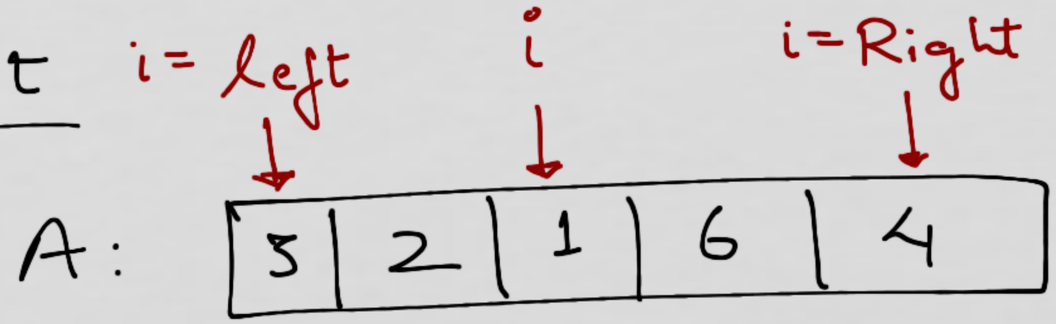
```
range([start], stop, [step])
range(3, 8)=[3,4,5,6,7]
range(3, 8, 2)=[3,5,7]
```

example:

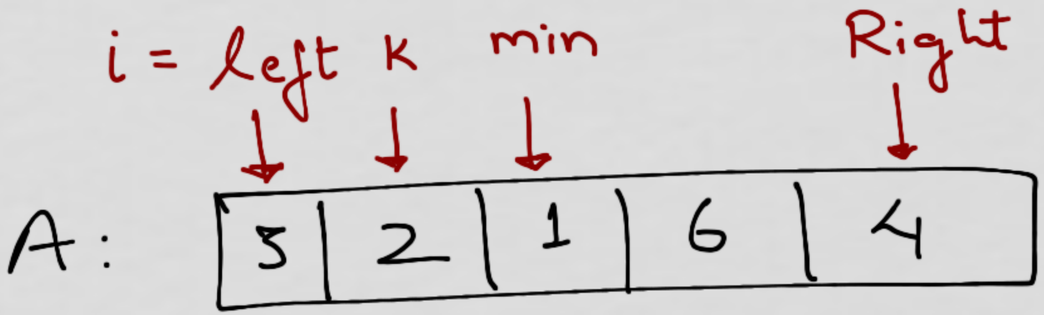
```
l = [10, 20, 30, 40]
sum=0
for index in range(len(l)):
    sum=sum+l[index]
print(sum)
```

Eg: Using Arrays & Loops

Selection Sort



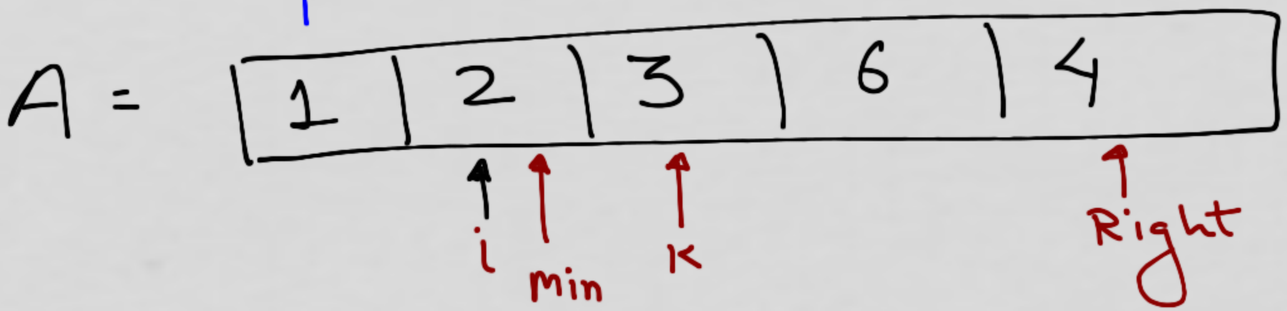
Strategy: Search the minimum value in the array and place it at the start of the array



Initially : i = left, unsorted

Finally : i = right, sorted

Swap A[i] and A[min]



Example

A=[5,2,1,6,4], left=0 right=4

[5,2,1,6,4] i=0 min=1 at position 2 => swap A[2] and A[i]

[1,2,5,6,4] i=1 min is 2 at position 1 => no need to swap

[1,2,5,6,4] i=2 min is 4 at position 4 => swap A[2] and A[i]

[1,2,4,6,5] i=3

[1,2,4,5,6] i=4

Selection Sort

```
def SelSort(A,left,right):
    for i in range(left, right):
        # INV  left<=i<right  A[left:i]  is a sorted list in ascending order and is
        #           less than all elements in remaining list A[i:right])
        min=i
        for k in range(i+1,right):
            #INV that A[min] is the minimum of A[i:k]
            if A[k] < A[min]:
                min=k
        # Finally A[min] is the minimum of A[i:right]
        A[i],A[min]=A[min],A[i]
```

```
l=[2,9,3,5,66,1]

SelSort(l,0,len(l))

print(l)

[1, 2, 3, 5, 9, 66]
```

```
def SelSort(A, left, right):
    i = left
```

assert: A[left, ..., right] is established

```
    while (i <= right):
        min = i
        k = i+1
```

assert: i ≤ min ≤ right, i+1 ≤ k ≤ right+1, min s.t. A[min] ≤ all A[i, ..., k-1]

```
        while (k <= right):
            if (A[k] < A[min]):
                min = k
            k = k+1
```

assert: ∃ min ∈ [i, right] s.t. A[min] ≤ all A[i, ..., right]

```
        A[i], A[min] = A[min], A[i]
        i = i+1
```

assert: A[left, ..., right] is sorted

assert: i ≤ left ≤ right+1, A[left, ..., i-1] is sorted

Slection Sort : Complexity

- Space?
- Time?

Slection Sort : Complexity

- Space?
  - O(1) constant space beyond the space for original list
- Time?
  - finding minimum is linear so n + (n-1) + (n-2) …+1 = $O(n^2)$