Python lists

Lists in Python are implemented as dynamic arrays. Their elements behave in the same way as Python variables do in general: **For elementary data types such as numbers, they contain the value**.

```
In [1]:
1  x = list(range(7))
2  y = x[2: 4]
3  y[0] = 7
4
5  print("x = ", x)
6  print ("y = ", y)

x =  [0, 1, 2, 3, 4, 5, 6]
y =  [7, 3]
```

When a sublist x[i: j] is created from x, **all the sublist elements are copied**.

Python lists

Lists in Python are implemented as dynamic arrays. Their elements behave in the same way as Python variables do in general: For elementary data types such as numbers, they contain the value.

```
In [1]:
1  x = list(range(7))
2  y = x[2: 4]
3  y[0] = 7
4
5  print("x = ", x)
6  print ("y = ", y)

x =  [0, 1, 2, 3, 4, 5, 6]
y =  [7, 3]
```

This takes O(j – i) time and space; in typical cases, that is O(*n*).

When a sublist x[i: j] is created from x, **all the sublist elements are copied**.

Python lists

Lists in Python are implemented as dynamic arrays. Their elements behave in the same way as Python variables do in general: For elementary data types such as numbers, they contain the value, **otherwise they are object references**.

```
In [1]:
1 x = list(range(7))
2 y = x[2: 4]
3 y[0] = 7
4
5 print("x = ", x)
6 print ("y = ", y)

x =  [0, 1, 2, 3, 4, 5, 6]
y =  [7, 3]
```

```
In [2]:
1 x = [[i] for i in range(7)]
2 y = x[2: 4]
3 y[0] = [7]
4
5 print("x = ", x)
6 print ("y = ", y)

x =  [[0], [1], [2], [3], [4], [5], [6]]
y =  [[7], [3]]
```

When a sublist x[i: j] is created from x, **all the sublist elements are copied**.

Python lists

Lists in Python are implemented as dynamic arrays. Their elements behave in the same way as Python variables do in general: For elementary data types such as numbers, they contain the value, **otherwise they are object references**.

```
In [1]:
1 x = list(range(7))
2 y = x[2: 4]
3 y[0] = 7
4
5 print("x = ", x)
6 print ("y = ", y)

x =  [0, 1, 2, 3, 4, 5, 6]
y =  [7, 3]
```

```
In [3]:
1 x = [[i] for i in range(7)]
2 y = x[2: 4]
3 y[0].pop()
4 y[0].append(7)
5
6 print("x = ", x)
7 print ("y = ", y)

x =  [[0], [1], [7], [3], [4], [5], [6]]
y =  [[7], [3]]
```

When a sublist x[i: j] is created from x, **all the sublist elements are copied**.

Tutorial 1.1 problem: Return the maximum

fastest iterative code

```
def max_iterative(listA):
    current_max_val = listA[0]
    for i in listA:
        if i > current_max_val:
            current_max_val = i
    return current_max_val
```

(by Chris Pickup)

fastest recursive code

```
def largestRecur(list, n):

    if n == 1:
        return list[n-1]
    else:
        previous = largestRecur(list, n-1)
        current = list[n-1]
        if previous > current:
            return previous
        else:
            return current
```

(by Sam Hardy)

Tutorial 1.1 problem: Return the maximum

fastest iterative code

```
def max_iterative(listA):
    current_max_val = listA[0]
    for i in listA:
        if i > current_max_val:
            current_max_val = i
    return current_max_val
```

(by Chris Pickup)

Both implementations run in O(*n*) time. The iterative code is more efficient by a factor 7.

fastest recursive code

```
def largestRecur(list, n):

    if n == 1:
        return list[n-1]
    else:
        previous = largestRecur(list, n-1)
        current = list[n-1]
        if previous > current:
            return previous
        else:
            return current
```

(by Sam Hardy)

Tutorial 1.1 problem: Return the maximum

$O(n^2)$ recursive code

```
def largestRecur(list):
    n = len(list)
    if n == 1:
        return list[n-1]
    else:
        previous = largestRecur(list[0: n-1])
        current = list[n-1]
        if previous > current:
            return previous
        else:
            return current
```

$O(n)$ recursive code

```
def largestRecur(list, n):

    if n == 1:
        return list[n-1]
    else:
        previous = largestRecur(list, n-1)
        current = list[n-1]
        if previous > current:
            return previous
        else:
            return current
```

(by Sam Hardy)

Sublist creation takes $O(n)$ time (and space)!