UPPSALA UNIVERSITET

MA3: Linked Lists

Tom Smedsaas

In this lecture we will discuss basic techniques for handling *linked lists*.
It covers the first 11 pages in the MA3.pdf document i.e. up to but not including “Iterators and generators”.

UPPSALA UNIVERSITET

Python lists

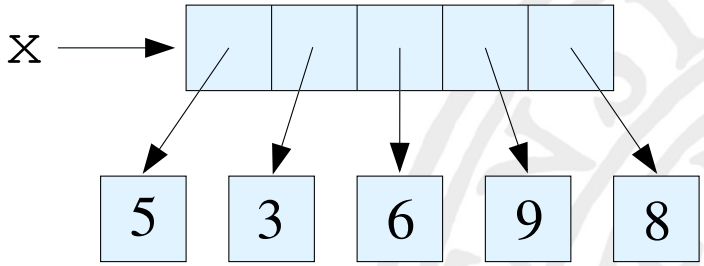
Lists are the most fundamental structure in Python.

We can, for example, write

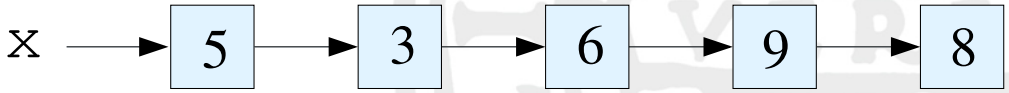
```
x = [5, 3, 6, 9, 8]
```

which can be illustrated:

Remember that it really is more like this:



We are now going to make a *linked list*:



UPPSALA UNIVERSITET

Why?

Why are we making an alternative to the Python `list`?

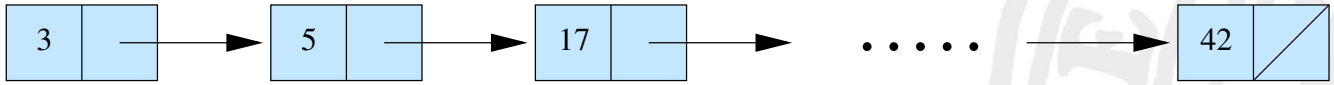
- Study basic techniques for handling linked structures.
- Study *iterators*, *generators* and *operator overloading*.
- See more examples of recursive methods.
- Practice algorithm analysis.
- Discover that there are situations where these lists are more efficient than the built in lists.

UPPSALA UNIVERSITET

A linked list

- We shall create a class `LinkedList` that can store a number of data items.
- The data shall be stored in *increasing order* so they must be comparable.
- We will use integers in our examples.

A linked list can be illustrated like this:



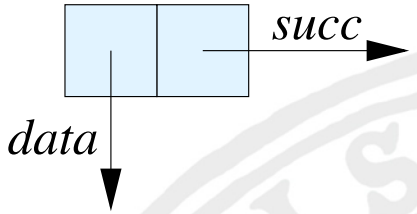
Each element has a reference to its follower.

UPPSALA UNIVERSITET

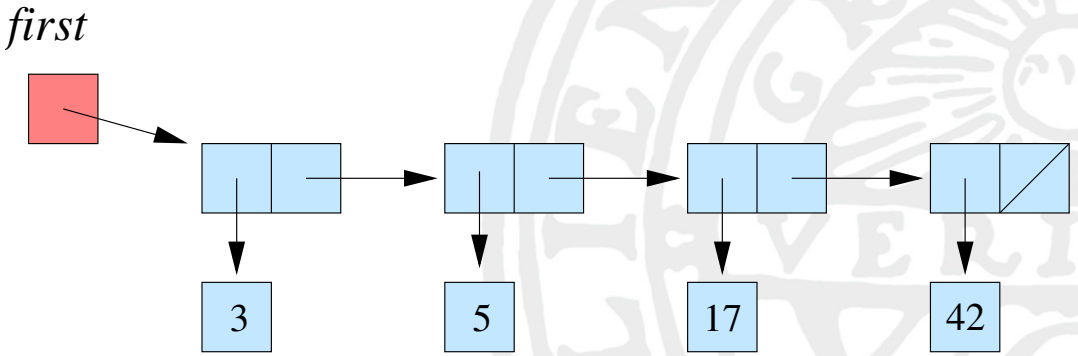
Python representation

We define a class for the nodes in the list:

```
class Node:
    def __init__(self, data, succ):
        self.data = data
        self.succ = succ
```



We have to keep track of the first node:



UPPSALA UNIVERSITET

A class for linked lists

We would like to be able to write like this:

```
ll = LinkedList()
print(ll)
for x in [5, 2, 3, 7]:
    ll.insert(x)
    print(ll)
```

```
()
(5)
(2, 5)
(2, 3, 5)
(2, 3, 5, 7)
```

Sketch:

```
class LinkedList:
    def __init__(self):
        pass

    def __str__(self):
        pass

    def insert(self, data):
        pass
```

We will also write the methods `remove_first` and `get_last` as examples of simple methods.

UPPSALA UNIVERSITET

The LinkedList class

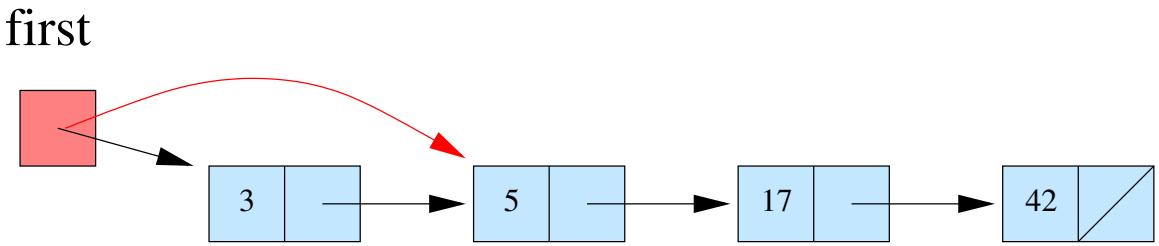
```
class LinkedList:

    class Node:
        def __init__(self, data, succ):
            self.data = data
            self.succ = succ


    def __init__(self):
        self.first = None

    . . .
```

The `Node` class is inside the `LinkedList` class

Creates an empty list

UPPSALA UNIVERSITET

The `remove_first` method




```
def remove_first(self):
    '''Removes the first element and returns its value'''
    result = self.first.data
    self.first = self.first.succ
    return result
```

There is actually a problem in the code. What is that?

What happens if the list is empty?

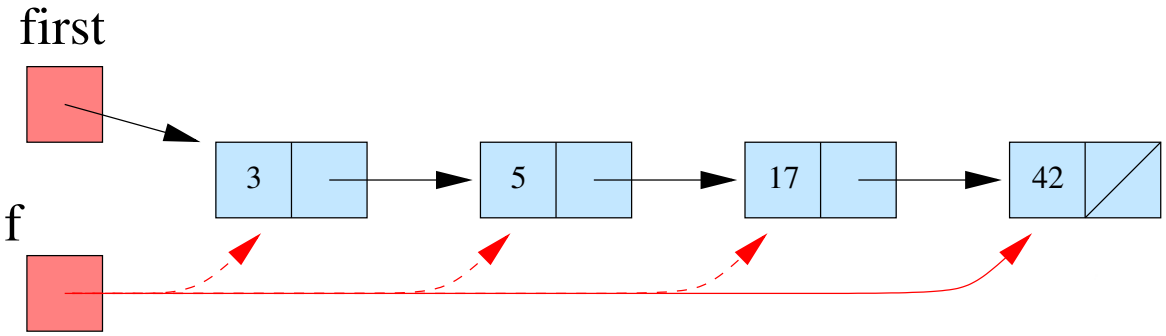
UPPSALA UNIVERSITET

Better `remove_first`

```
def remove_first(self):
    '''Removes the first element and returns its value'''
    if self.first == None:
        return None                 # Or raise an exception
    result = self.first.data
    self.first = self.first.succ
    return result
```

UPPSALA UNIVERSITET

The `get_last` method




```
def get_last(self):
    '''Returns the last stored data item'''
    if self.first == None:
        return None                 # Or raise an exception
    f = self.first
    while f.succ:
        f = f.succ
    return f.data
```

UPPSALA UNIVERSITET

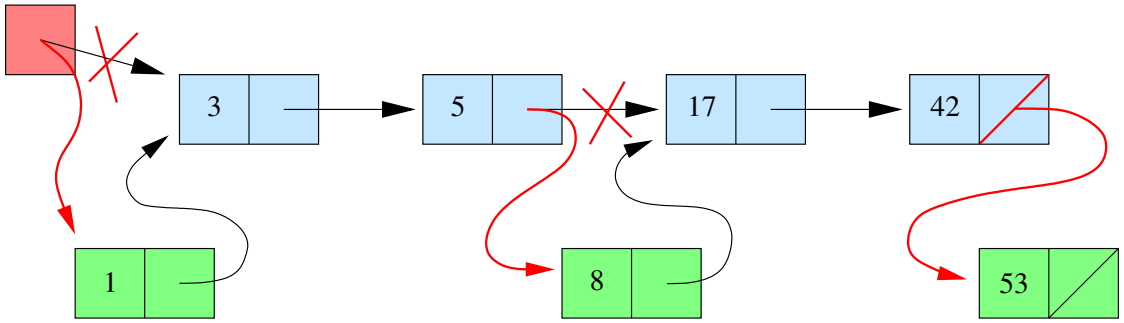
The __str__ method

```
def __str__(self):
    result = ''
    f = self.first
    while f:
        result += str(f.data)
        f = f.succ
        if f:
            result += ', '
    return '(' + result + ')'
```

Comma *between* elements

UPPSALA UNIVERSITET

An *iterative* `insert` method




```
    def insert(self, x):
        if self.first is None or x < self.first.data:
            self.first = self.Node(x, self.first)
        else:
            f = self.first
            while f.succ and x >= f.succ.data:
                f = f.succ
            f.succ = self.Node(x, f.succ)
```

UPPSALA UNIVERSITET

A *recursive* insert: method 1.

In the `Node` class:

```
def insert(self, x):
    if self.succ is None or x < self.succ.data:
        self.succ = self.Node(x, self.succ)
    else:
        self.succ.insert(x)
```

and the `LinkedList` class:

```
def insert(self, x):
    if self.first is None or x < self.first.data:
        self.first = self.Node(x, self.first)
    else:
        self.first.insert(x)
```

UPPSALA UNIVERSITET

A *recursive* insert: method 2.

With a *recursive help method* in `LinkedList`

```
def insert(self, x):
    self.first = self._insert(x, self.first)

def _insert(self, x, f):
    if f is None or x < f.data:
        return self.Node(x, f)
    else:
        f.succ = self._insert(x, f.succ)
        return f
```

UPPSALA UNIVERSITET

The end