MA3: Some Python facilities

This lecture discusses

- *iterators*,
- *generators* and
- *operator overloading*

applied to the `LinkedList` class

A common pattern

Suppose we want to

- sum all values in our linked list or
- compute the mean and standard deviation of the values in the list or
- find the first prime number in the list or
- …

For ordinary lists this could be done with a for-statement and we would like to be able to it for our linked list also

A for-loop for the linked lists

```
ll = LinkedList()
. . .  # Build up the list

sum = 0
n = 0
for x in ll:
    sum += x
    n += 1
print(f'The mean value is {sum/n}')
```

This code is written *without any knowledge of the internal structure* of `LinkedList` class!

Make the list *iterable*!

Add two special methods:

```
def __iter__(self):
    self.current = self.first
    return self

def __next__(self):
    if self.current:
        result = self.current.data
        self.current = self.current.succ
        return result
    else:
        raise StopIteration
```

Now we can iterate over our lists with a `for` statement

An easier way

We can write the __iter__ method as a *generator* :

```python
def __iter__(self):
    current = self.first
    while current:
        yield current.data
        current = current.succ
```

Note:

- The yield statement
- No __next__ method
- current is a local variable

Operator overloading

By operator overloading we mean to give existing operators like +, ==, <=, … a meaning and definition for new data types.

For an ordinary list we can use the operator `in` for example in an expression like

```
if w in lista:
    print(’Oui!’)
else:
    print (’Non!’)
```

We can get this to work by implementing the `__in__` method in our `LinkedList` class.

In the LinkedList class

```
def __in__(self, x):
    for d in self:          # Use generator/iterator
        if x == d:
            return True
        elif x < d:         # No point in searching more
            return False
    return False
```

Another example: indexing

```
def __getitem__(self, index):
    i = 0
    for x in self:
        if i == index:
            return x
        i += 1
    raise IndexError(f'LinkedList index {index} out of range’)
```

Now we can write code like:

```
print(ll[0] + ll[2])
```

but **not**:

```
ll[3] = ll[0] + ll[2])
```

UPPSALA UNIVERSITET

The end