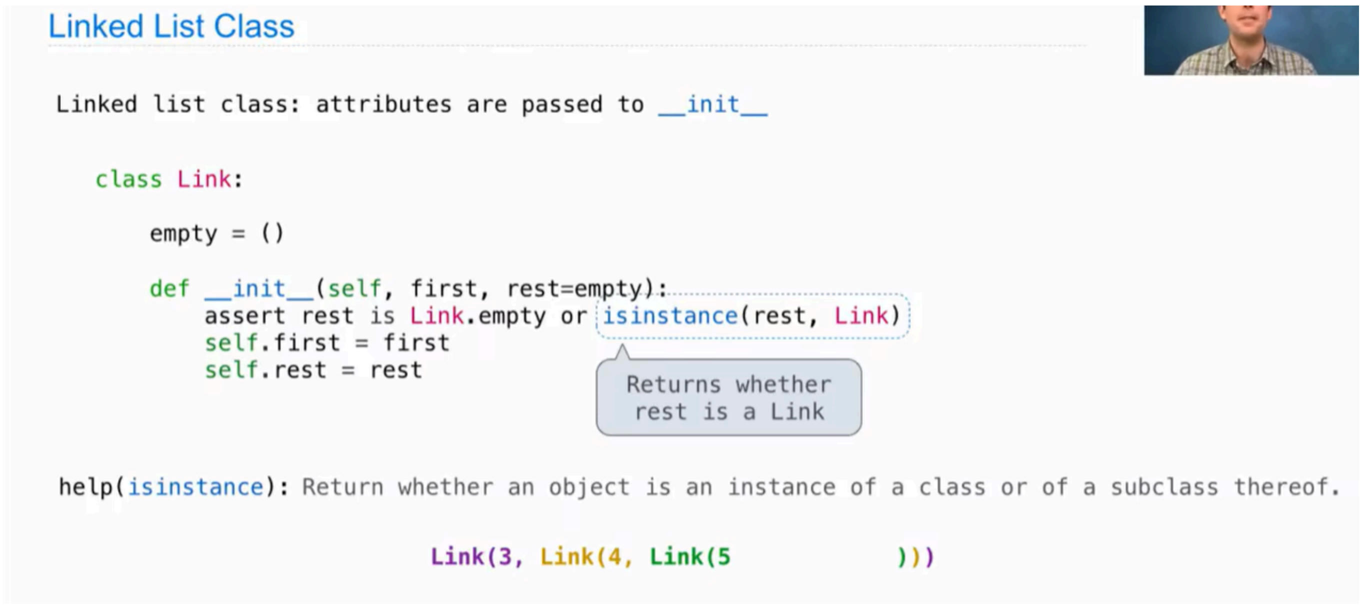
Lecture 21.Composition

Linked Lists

implementation

```
class Node:

    def __init__(self, value, next: Node):
        self.value = value
        self.next = next
```



Linked List Class

Linked list class: attributes are passed to __init__

```
class Link:

    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
```

Returns whether rest is a Link

help(isinstance): Return whether an object is an instance of a class or of a subclass thereof.

Link(3, Link(4, Link(5)))

Linked List Processing

```
class Link:

    empty = ()

    def __init__(self, first, rest):
        self.first = first
        self.rest = rest
```

we know that we have built-in function:

- square
- map

- filter

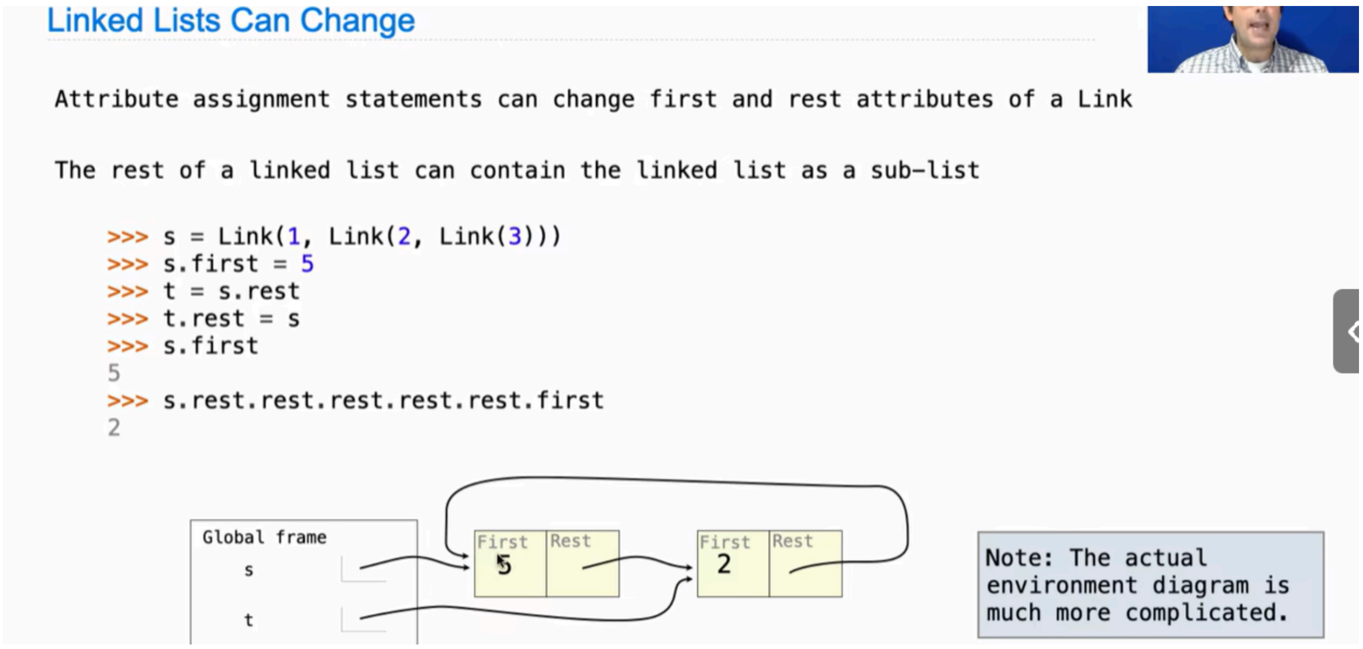
  but we can't apply them directly to linked list

```
def range_link(start, end)
    """return a link containing consecutive intergers from start to end"""
    if start >= end:
        return Link.empty
    else:
        return Link(star, range_link(start + 1, end))

def map_link(f, s):
    """return a link that contains f(x) fro each x in link s"""
    if s is Link.empty:
        return s
    else:
        return Link(f(s.first), map_link(f, s.rest))

def filter(f, s)
""" retrun a link that contains only the elments x of link s for which f(x) is
a true value"""
    if s is Link.empty:
        return s
    filtered_rest = filter_link(f, s.rest)
    if f(s.first):
        return Link(s.first, filtered_rest)
    else:
        return filtered_rest
```

Linked Lists Mutations



Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.rest.first
2
```

Note: The actual environment diagram is much more complicated.

Linked List Mutation Example

```
def add(s, v):
    """add v to an orderd list s with no repeats, returning modified s."""
    assert s is not List.empty
    if s.fisrt > v:
        s.first, x.rest = v, Link(s.first, s.rest)
    elif s.first < v and empty(s.rest):
        s.rest = Link(v)
    elif s.rest < v:
        add(s.rest, v)
    return s
```

Tree Class

```
class Tree:
    def __init__(self, label, barnches=[])
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

    def __repr__(self):
        if self.branches:
            branch_str = ', ' + repr(self.branches)
        else:
            branch_str = ' '
        return 'Tree({0}{1})'.format(repr(self.label), branch_str)

    def __str__(self):
        return '\n'.join(self.indented())

    def indented(self):
        lines = []
        for b in self.branches:
            for line in b.indented():
                lines.append('    ' + line)
        return [str(self.label)] + lines

    def is_leaf(self):
        return not self.branches


def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])

def leaves(t):
    if t.is_leaf():
        return [t.label]
    else:
        all_leaves = []
        for b in t.branches:
            all_leaves.extend(leaves(b))
        return all_leaves

def height(t):
    if t.is_leaf():
        return 0
    else:
        return 1 + max(height(b) for b in t.branches)
```

Example Pruning Trees

Removing subtress from a tree is called pruninng

Prune branches befor recursive processing.

```
def prune(t, n):
    """prune all sub-trees whose label is n"""
    t.branches = [b for b in t.branches if b.label != n]
    for b in t.branches:
        prune(b, n)
```