# CS61A Lecture 23

Monday, October 21st, 2019

## Announcements

- Midterm study guide is out. Includes minor revision to MT1 guide.
- No lecture on Wednesday, and video-only lecture on Friday by Allan Kaye, who created the first OOP language.
- No discussion this week.

## Data examples

We're just gonna review today with some old exam questions, and revise some commonly forgotten concepts.

### Lists

Lists can have names assigned to them through environment diagrams, and they are mutable.

Assume we start with the following code before example:\

```
s = [2,3]
t = [5,6]
```

| Operation | Example | Result |
|---|---|---|
| `append` adds one element to a list. | `s.append(t)`<br>`t = 0` | `s = [2,3, [5,6]]`<br>`t=0` |
| `extend` adds all elements in one list to another list.<br>( `extend` does the same thing as `+=` ) | `s.extend(t)`<br>`t[1] = 0` | `s = [2,3,5,6]`<br>`t = [5,0]` |
| addition and slicing create new lists containing existing elements | `a = s + [t]`<br>`b = a [1:]`<br>`a[1] = 9`<br>`b[1][1] = 0` | `s = [2,3]`<br>`t = [5,0]`<br>`a = [2, 9, [5, 0]]`<br>`b = [3, [5, 0]]` |
| `list` function also creates a new list containing existing elements | `t = list(s)`<br>`s[1] = 0` | `s = [2,0]`<br>`t = [2,3]` |
| slice assignment replaces a slice with new values | `s[0:0] = t`<br>`s[3:]=t`<br>`t[1]=0` | `s = [5,6,2,5,6]`<br>`t = [5,0]` |

| Operation | Example | Result |
|---|---|---|
| `pop` removes and returns and the last element | `t = s.pop()` | `s = [2]`<br>`t = 3` |
| `remove` removes the first element equal to the argument | `t.extend(t)`<br>`t.remove(5)` | `s = [2,3]`<br>`t = [6,5,6]` |
| slice assignment can remove elements from a list by assigning to an empty slice | `s[:1] = []`<br>`t[0:2] = []` | `s = [3]`<br>`t = []` |

> **What happens when you append lists to each other?**
> Say we ran the following code.
>
> ```
> >>> s, t = [2,3], [5,6]
> >>> s.append(t)
> >>> t.append(s)
> >>> print(s)
> [2,3,[5,6,[...]]]
> >>> s[2][2][2][2][2][1]
> 6
> ```

# Objects

Instance attributes are found before class attributes, and class attributes are inherited. Check the slides for a more detailed example.

# Linked Lists

The attributes of a linked list can be changed. Let's review it with the `Link` class we defined before.

```
class Link:
    empty = ()

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

    def __getitem__(self,i):
        if i == 0:
            return self.first
        else:
            return self.rest

    def __len__(self):
        return 1 + len(self.rest)

    def __repr__(self):
        if self.rest:
            rest_str = ', ' + repr(self.rest)
        else:
```

```
            rest_str = ''
        return 'Link{0}{1}'.format(self.first,rest_str)
```

Attribute assignment statements can change the first and rest attributes of a Link, and the rest of a linked list can contain the linked list itself as a sublist.

```
>>> 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
```

# Set Mutation

We can also implement operations that change a set. Let's say we have an ordered linked list `u` .

```
>>> u = Link(1,Link(3,Link(5)))
```

We want to implement a function `add` , different from `join` , because we want to mutate `u` to be a set that includes the number `v` , while join would create a new set that doesn't affect `0` .

```
add(u,0)
```

One way to do this is to edit the original instance of 1 to 0, create a new instance of Link with `first` as 1, and then re-bind this to be the rest of the link.

```
add(u,3)
```

We don't do anything because we realize 3 is already in `u` .

```
add(u,4)
```

We move along the list until we find a value bigger than 4, edit the value to 4, and create a new instance with the original value of that instance.

```
add(u,6)
```

We move along the list until we get to the end and nothing is smaller than the value `v` , thus, we add to the end of the linked list.

How can we implement this function?

```
def add(s,v):
    assert not empty(s)
    if s.first > v:
```

```
        s.first, s.rest = __, __
    elif s.first < v and empty(s.rest)
        ____
    elif s.first < v:
        ____
    return s

## SOLUTION ##

def add(s,v):
    assert not empty(s)
    if s.first > v:
        s.first, s.rest = v, Link(s.first, s.rest)
    elif s.first < v and empty(s.rest):
        s.rest = Link(v,s.rest)
    elif s.first < v:
        add(s.rest,v)
    return s
```