Operations on lists

- Iteration

```
for e in Univs:
    print('Univs contains ')
    print(e)
    print('    which contains')
    for u in e:
        print('        ' + u)
```

Append versus flatten

```
Techs.append(Ivys)
```

Side Effect

Then `Techs` returns

```
['MIT', 'Cal Tech', 'RPI',
  ['Harvard', 'Yale', 'Brown']]
```

```
flat = Techs + Ivys
```

Creates a new list

Then `flat` returns

```
['MIT', 'Cal Tech',
  'RPI','Harvard', 'Yale', 'Brown']
```

In more detail

```
>>>Techs
['MIT', 'Cal Tech',
  'RPI']

>>>Techs.append(Ivys)

>>>Techs
['MIT', 'Cal Tech',
  'RPI', ['Harvard',
  'Yale', 'Brown']]
```

Mutating

```
>>>Techs
['MIT', 'Cal Tech',
  'RPI']

>>>flat = Techs + Ivys

>>>flat
['MIT', 'Cal Tech',
  'RPI', 'Harvard',
  'Yale', 'Brown']

>>>Techs
['MIT', 'Cal Tech',
  'RPI']
```

Creating

Cloning

- Avoid mutating a list over which one is iterating
- Example:

```
def removeDups(L1, L2):
    for e1 in L1:
        if e1 in L2:
            L1.remove(e1)
```

```
L1 = [1,2,3,4]
L2 = [1,2,5,6]
removeDups(L1, L2)
```

Then

```
print(L1)
```

returns

```
[2, 3, 4]
```

Why?

```
def removeDups(L1, L2):
    for e1 in L1:
        if e1 in L2:
            L1.remove(e1)
```

- Inside for loop, Python keeps track of where it is in list using internal counter
- When we mutate a list, we change its length but Python doesn't update counter

Better is to clone

```
def removeDupsBetter(L1, L2):
    L1Start = L1[:]
    for e1 in L1Start:
        if e1 in L2:
            L1.remove(e1)
```

```
L1 = [1,2,3,4]
L2 = [1,2,5,6]
removeDupsBetter(L1,
  L2)
```

Then

```
print(L1)
```

returns

```
[3, 4]
```

Note that using `L1Start = L1` is not sufficient

L1[:]