

Lists

- Look a lot like tuples
 - Ordered sequence of values, each identified by an index
 - Use [1, 2, 3] rather than (1, 2, 3)
 - Singletons are now just [4] rather than (4,)
- **BIG DIFFERENCE**
 - Lists are mutable!!
 - While tuple, int, float, str are immutable
 - So lists can be modified after they are created!

Why should this matter?

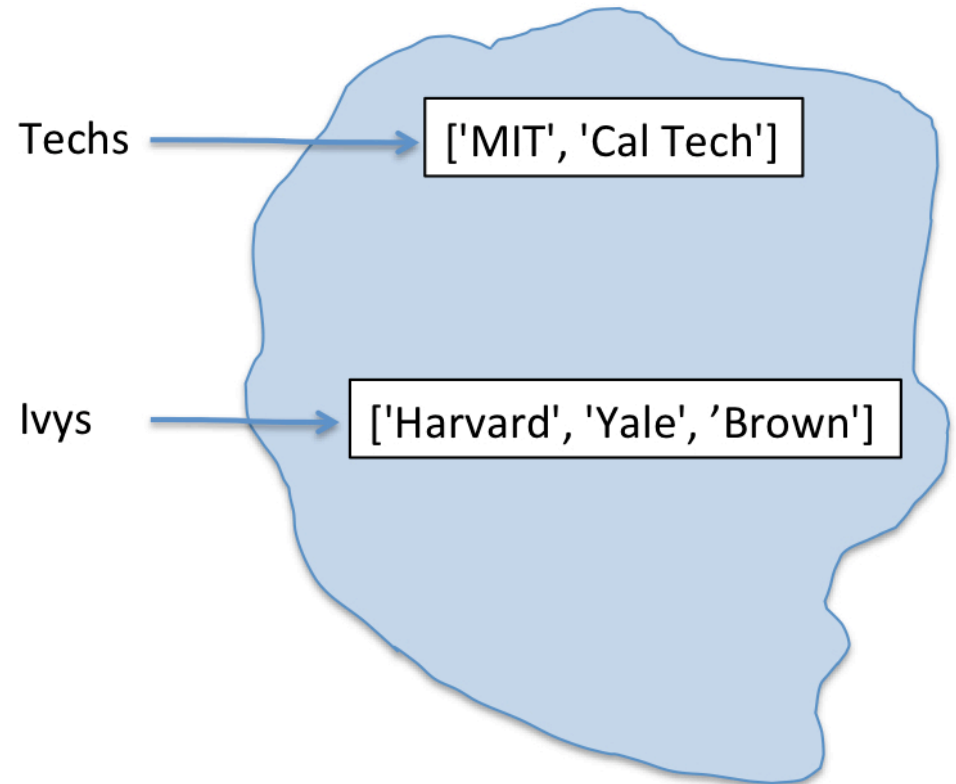
- Some data objects we want to treat as fixed
 - Can create new versions of them
 - Can bind variable names to them
 - But don't want to change them
 - Generally valuable when these data objects will be referenced frequently but elements don't change
- Some data objects may want to support modifications to elements, either for efficiency or because elements are prone to change
- Mutable structures are more prone to bugs in use, but provide great flexibility

Visualizing lists

```
Techs = [ 'MIT',  
          'Cal Tech' ]
```

```
Ivys = [ 'Harvard',  
         'Yale', 'Brown' ]
```

```
>>> Ivys[1]  
'Yale'
```



Structures of lists

- Consider

```
Univs = [Techs, Ivys]
```

```
Univs1 = [[ 'MIT', 'Cal Tech' ],  
          [ 'Harvard', 'Yale', 'Brown' ]]
```

- Are these the same thing?
 - They print the same thing
 - But let's try adding something to one of these

Mutability of lists

- Let's evaluate

```
Techs.append( 'RPI' )
```

- Append is a method (hence the .) that has a **side effect**
 - It doesn't create a new list, it mutates the existing one to add a new element to the end
- So if we print Univs and Univs1 we get different things

```
print(Univs)
```

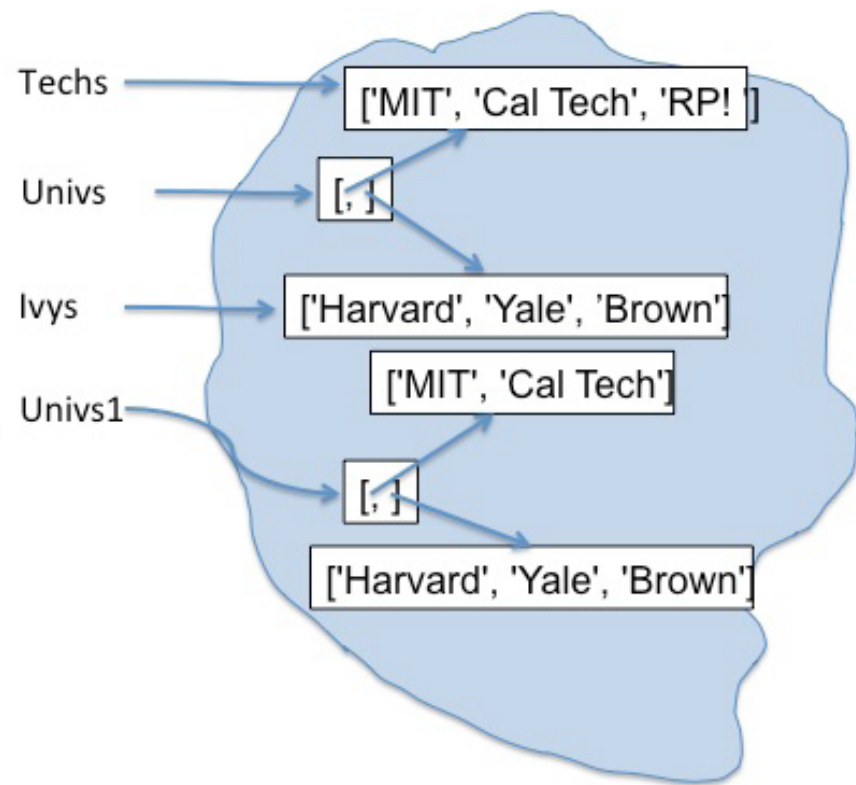
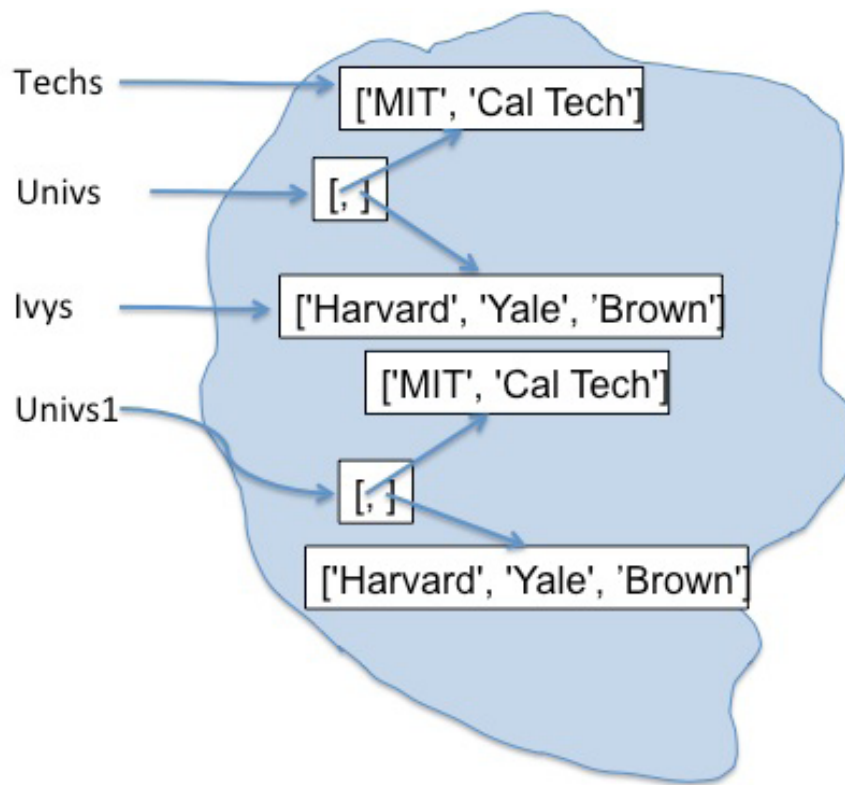
```
Univs = [['MIT', 'Cal Tech',  
         'RPI'], ['Harvard', 'Yale',  
         'Brown']]
```

```
Print(Univs1)
```

```
Univs1 = [['MIT', 'Cal Tech'],  
          ['Harvard', 'Yale', 'Brown']]
```

Why?

- Bindings before append
- Bindings after append



Observations

- Elements of `Univs` are not copies of the lists to which `Techs` and `Ivys` are bound, but are the lists themselves!
- This effect is called **aliasing**:
 - There are two distinct paths to a data object
 - One through the variable `Techs`
 - A second through the first element of list object to which `Univs` is bound
 - Can mutate object through either path, but effect will be visible through both
 - Convenient but **treacherous**

We can directly change elements

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

```
>>> Techs[2] = 'WPI'
```

Cannot do this with tuples!

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