Lists
列表

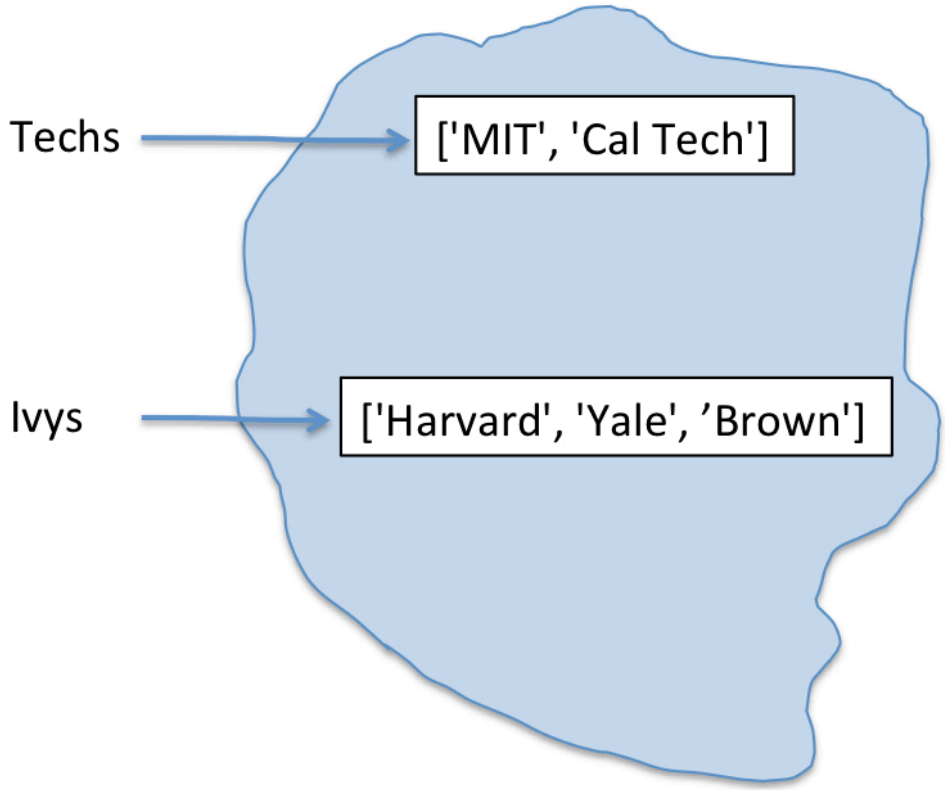
- Look a lot like tuples 元组
  - Ordered sequence of values, each identified by an index
  - Use [1, 2, 3] rather than (1, 2, 3)
  - Singletons are now just [4] rather than (4,)
单元素
- **BIG DIFFERENCE**
  - Lists are mutable!! 列表是可以改变的
  - While tuple, int, float, str are immutable
而元组，整数，浮点数，字符串是不可以改变的
  - So lists can be modified after they are created!
列表可以在创作后被改变

Why should this matter?

- Some data objects we want to treat as fixed
  - 有些数据对象我们想将它看作是固定的。
  - Can create new versions of them
  - Can bind variable names to them
  - 可以将变量名称捆绑给它
  - But don't want to change them
  - 但是不想改变它们
  - Generally valuable when these data objects will be referenced frequently but elements don't change
  - 通常当这些数据对象被频繁的参考但是元素不发生改变时是有价值的。
- Some data objects may want to support modifications to elements, either for efficiency or because elements are prone to change
- 易于改变
- Mutable structures are more prone to bugs in use, but provide great flexibility
  - 可变结构在使用中更容易出现bugs，但是提供了很大的灵活性。

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
列表的可变化性

- Let's evaluate

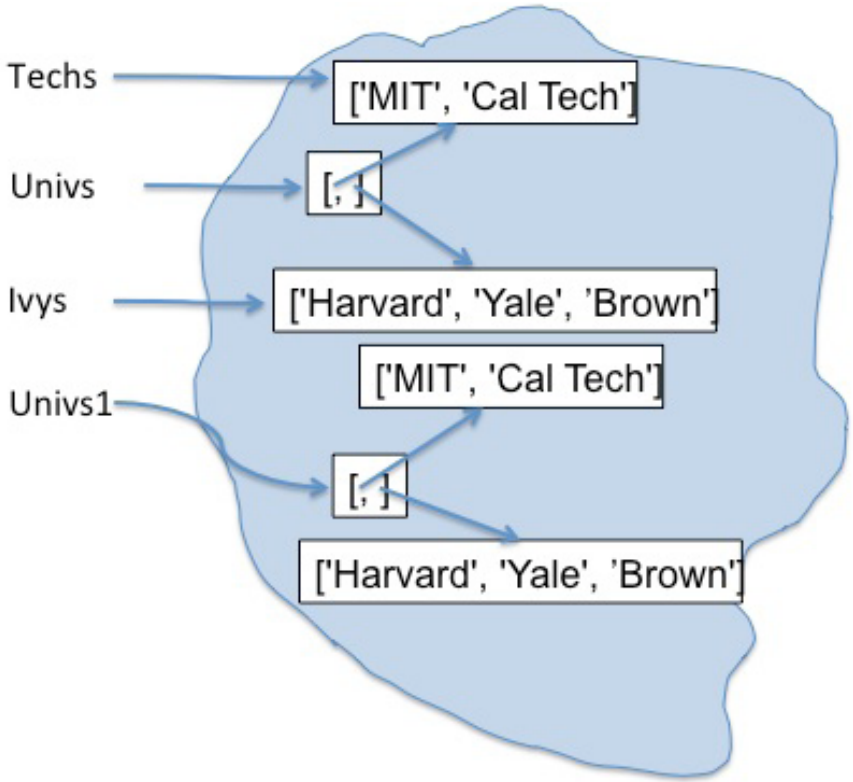
```
Techs.append('RPI')
```

- Append is a method (hence the .) that has a **side effect** 副作用
  - 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

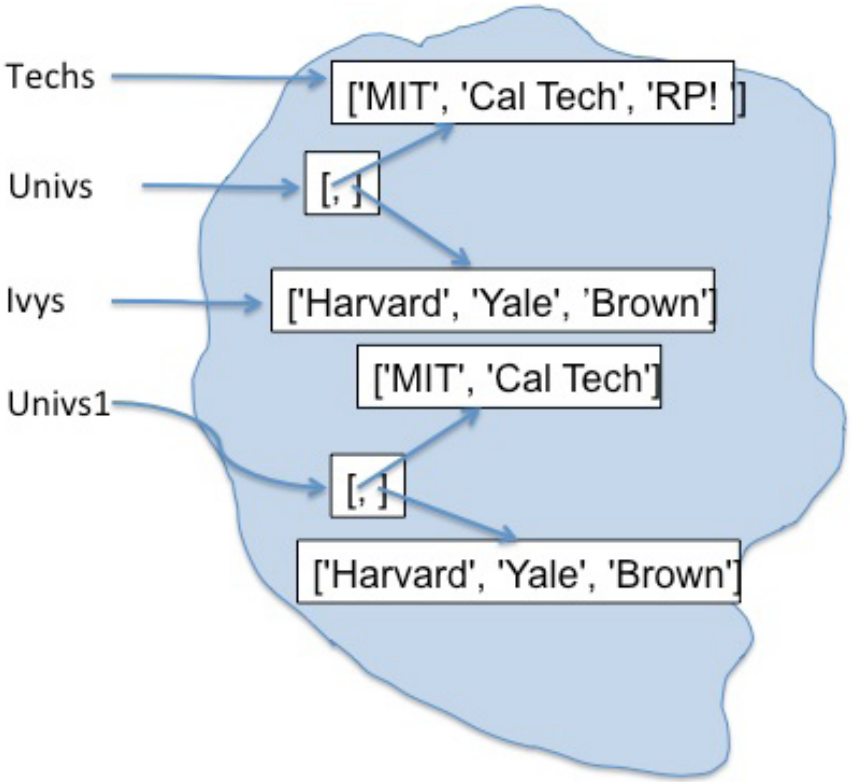
```
print(Univs)
Univs = [['MIT', 'Cal Tech',
  'RPI'], ['Harvard', 'Yale',
  'Brown']]

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

Why?

- Bindings before append
  边界，封皮



- 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**

化名，同义名

这里有两个清晰的通路到底数据目标

一种是通过可变的Techs

另一种是univs捆绑的列表目标中的第一个元素

可变目标可以通过任意一个道路，但是影响可以从两个道路中看到。

方便的但是变幻莫测的

We can directly change elements

我们可以直接地改变元素

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

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

Cannot do this with tuples!

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