Announcements

- Homework
  - Homework 6 due tonight!
- No lab today, I'll just be around to help with homework for a bit if people have questions
- I'll be around until probably 6 or 7 tonight, but at Physics Tea at 3
- Polling: `rembold-class.ddns.net`

Review Question

The code below is run. How would you then access the `'ninja'` element in tuple `D`?

A) `D[1][0]`

B) `D[0][1]`

C) `D[0][2]`

D) `D[1][2]`

```
A = ('pirate', 'ninja')
B = ('samurai',) + A
C = (B, ('ship', 'rope', 'horse'))
D = C[::-1]
```

Solution: `D[1][2]`

Mutability: Part I

- Touched on mutability before in that strings and tuples are immutable
  - We can not do the below:

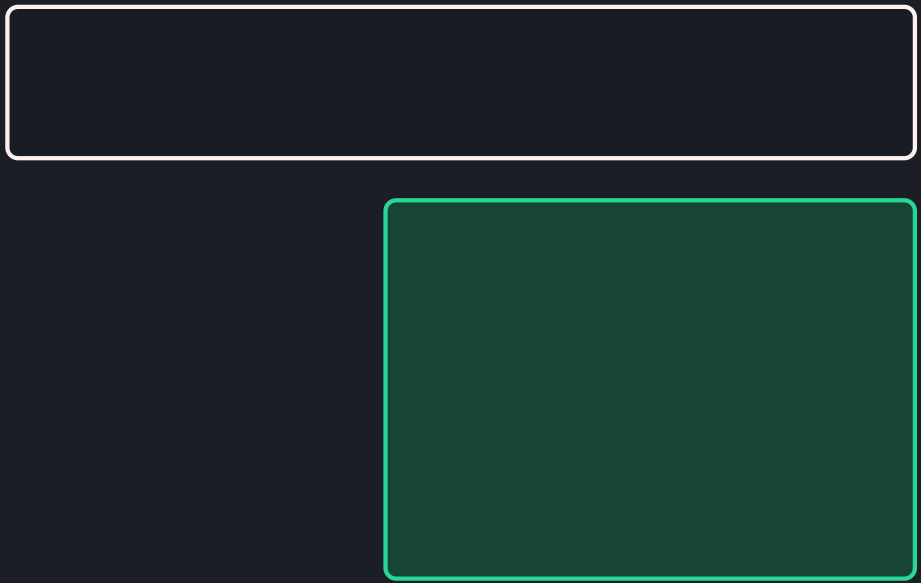
```
A = 'hello'
A[0] = 'H'

B = ('This', 'is', 'Sparta')
B[2] = 'Patrick'
```

Mutability: Part I

- Touched on mutability before in that strings and tuples are immutable
  - We can not do the below:

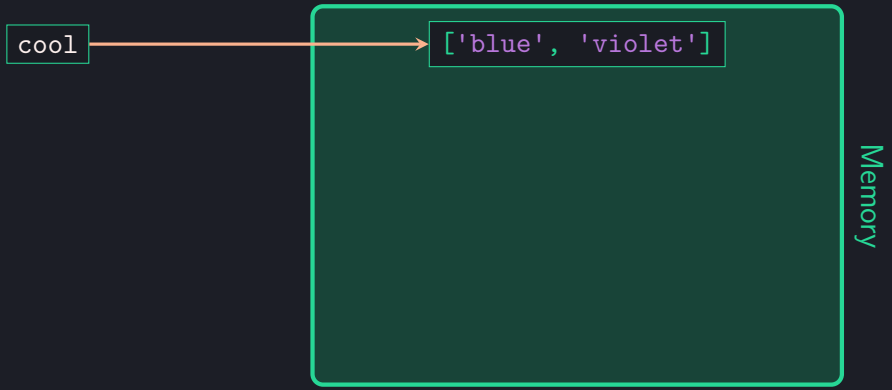
```
A = 'hello'
A[0] = 'H'

B = ('This', 'is', 'Sparta')
B[2] = 'Patrick'
```

- Presumably, this is allowed with lists (and it is)
- Mutability has some other ramifications though that we want to touch on


Code
Memory

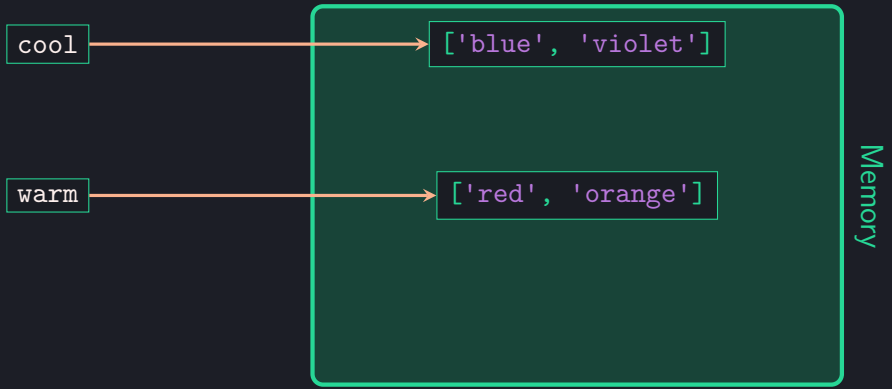
Code

```
cool = ['blue', 'violet]
```



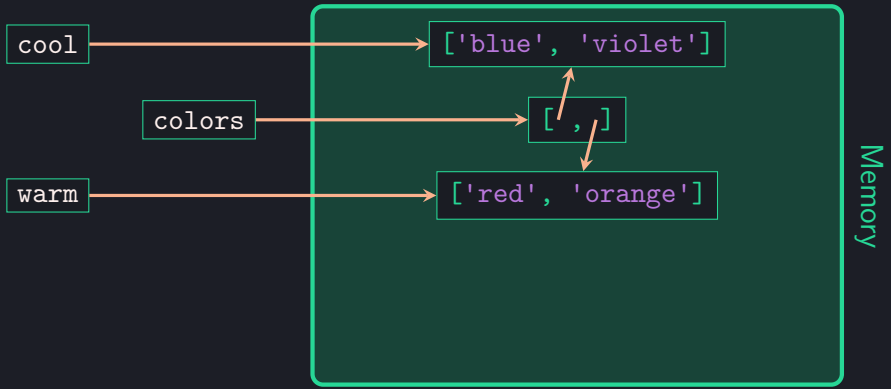
Code

```
warm = ['red', 'orange']
```



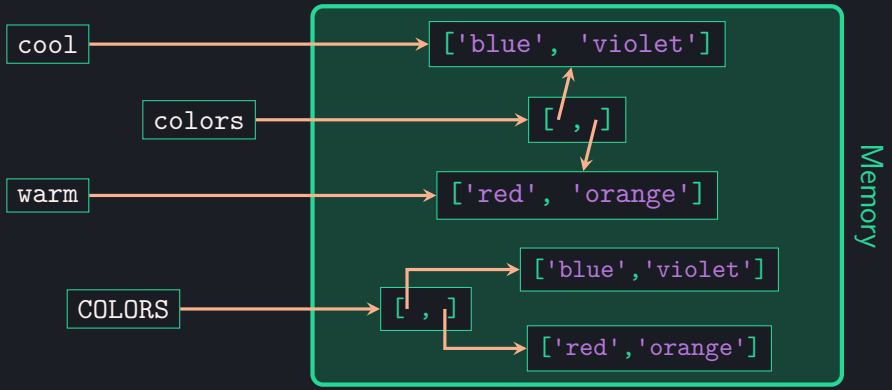
Code

```
colors = [cool, warm]
```



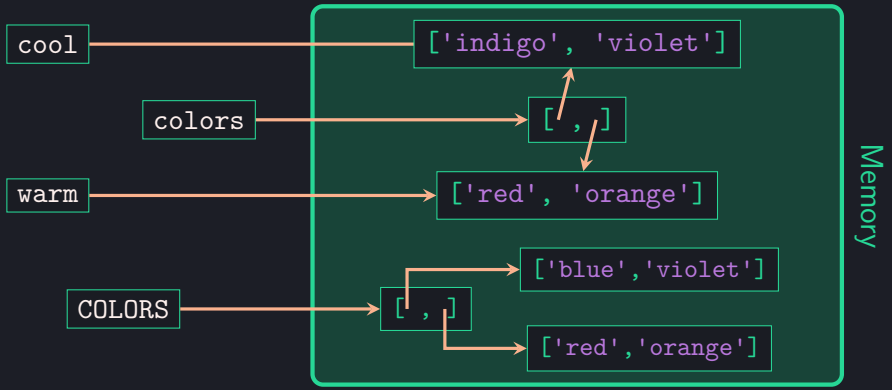
Code

```
COLORS = [['blue','violet'],['red','orange']]
```



Code

```
cool[0] = 'indigo'
```



Append, Extend, Comprehend

- Can add a new item to a list with `append` method

```
A = ['tuna']
A.append('Cod')
A.append('Halibut')
A.append('Salmon')
print(A)
```

- Can add many new items to list with `extend` method

```
A = ['tuna']
A.extend(['Cod', 'Halibut', 'Salmon'])
print(A)
```

More useful list methods

- Can remove things from lists easily by value
  - `L.remove(e)`
  - Removes the *first* occurance of `e` from the list `L`
- Can find values in the list
  - `L.index(e)`
  - Returns the index of the *first* occurance of `e` in the list `L`
- Can sort the list
  - `L.sort()`
  - Sorts the list in ascending order

Sneaky Mutability

- Mutability is usually great for its flexibility
- I've found two real instances when I have to be careful
  - Initializing a list to look like another list, wanting to make changes to one and then compare
    - Example
  - Looping over a mutating list
    - Example

Cloning

- What can we do then in these instances to not let mutability mess us up?
- Clone the list instead of just assigning a new variable to it!
  - Creates a *new object* in memory
  - Several different ways you can make a clone:
    - Slicing
    - Using `list`

Understanding Check

Given the code to the right, what would be the final printed value of A?

A) `['Fox', 'Giraffe', 'Hippo', 'Iguana']`

B) `['Fox', 'Hippo', 'Iguana']`

C) `['Iguana', 'Fox']`

D) `['Fox', 'Iguana']`

```
A = [
    'Fox',
    'Giraffe',
    'Hippo'
    ]
A.append('Iguana')
A[:].reverse()
B = A
for anim in B:
    if anim[1] == 'i':
        B.remove(anim)
print(A)
```

Solution: `['Fox', 'Hippo', 'Iguana']`