Announcements

- Homework
  - I got the HW4 graded. Hopefully will be able to start in on HW5 before this weekend.
  - You have HW6 due on Friday! Should be able to answer everything I think after today!
- CS Tea tomorrow at 11:30!
- Polling: `rembold-class.ddns.net`

Review Question

Which of the following is an invalid way of importing modules and referring to them?

A

```
import circles
y = circles.area(5)
print(y)
```

B

```
from circles import area
y = circles.area(5)
print(y)
```

C

```
from circles import *
y = area(5)
print(y)
```

D

```
import circles as c
y = c.area(5)
print(y)
```

Tuple Operations

- Almost identical to available string operations
  - Strings are basically just a special tuple
- Can add for concatenation (adding one to the end of another)
  - `(1,2,3) + ('a', 'b', 'c')`
- Can duplicate with multiplication
  - `3*(1,2,3)`
- Can index and slice just like strings
  - `(1,2,3,4)[2]`
    - Single index gives you back whatever variable type that element is
  - `(1,'b',3,'d')[:2]`
    - Slices return another tuple

Tuples all the way down

- Tuples can contain more tuples!
  - Getting through concatenation can be a bit tricky, so be careful

```
t1 = (1,2,3)
t2 = (t1, 2*t1, 'a', ('x','y','z'))
```

- Means in some cases you might need multiple indexes to "drill down" to get the value that you want
  - Nested indexes work from "outside in"

What are they good for?

- Can loop through tuples just like we could with strings
- Can either loop through the indices or the values directly:
  - Indices:

```
t = ('a', 'b', 'c')
for i in range(len(t)):
    print(t[i])
```

  - Values directly:

```
t = ('a', 'b', 'c')
for value in t:
    print(value)
```

Understanding Check

What would be the output of the printed statement in the code to the right?

A) `(1,'a','b')`

B) `(1,'a')`

C) Error: can't add strings and tuples

D) Error: index out of range

```
A = (1,3,5)
B = (2*A, ('a',))
C = B + ('b','c','d')
D = ()
for v in C[:2]:
    D += v[:1]
print(D)
```

Out of Control Slicing

- Try to slice a tuple (or string!) where one side of the slice would be out of range of the tuple
  - `(1,2,3)[1:100]`
- Instead of giving an error, Python will return what it thinks you wanted
  - Everything starting at 1 and up till the end of the list
  - `(1,2,3)[1:100] == (1,2,3)[1:3]`
- Works the same for negative indices as well!
- A slice will not give you an out-of-bounds error, it just returns what it can

Multiple Assignment

- We've already seen we can assign multiple variables at once
  - `x, y = 2, 5`
- We can use this same syntax to "unpack" tuples into separate variables
  - `row, col = (2,5)`
  - `x, y, z = ('fish', 'steak', 'potatoes')`
  - `(a,b,c) = (1,2,3)`
- You need the same number of variables as elements of your tuple for this to work
  - Assign dummy variables if you need (a common one is _)

Count 'em off

- We know how to use a `for` loop to loop over either indices or values
  - If indices, we can always get the value
  - If values though, we'd have to track the indices ourselves
- The values notation tends to make a lot of sense for many, so they prefer it
- Still have many situations where it is important to know an index of a value
  - Can use `enumerate`!

```
t = ('a', 'b', 'c')
for i,v in enumerate(t):
    print('Index:',i,'Value:', v)
```

`list` out the similarities. . .

- A `list` is another type of non-scalar object in Python
  - Delimited with square brackets: `A = [1,2,3]`

`list` out the similarities. . .

- A `list` is another type of non-scalar object in Python
  - Delimited with square brackets: `A = [1,2,3]`
- Very similar to a tuple, in that they are an ordered sequence
  - Still concatenate with addition
  - Still can index, and slice
  - Still can loop over with `for` loops

`list` out the similarities...

- A `list` is another type of non-scalar object in Python
  - Delimited with square brackets: `A = [1,2,3]`
- Very similar to a tuple, in that they are an ordered sequence
  - Still concatenate with addition
  - Still can index, and slice
  - Still can loop over with `for` loops
- The primary difference?
  - Lists are mutable!

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'
```

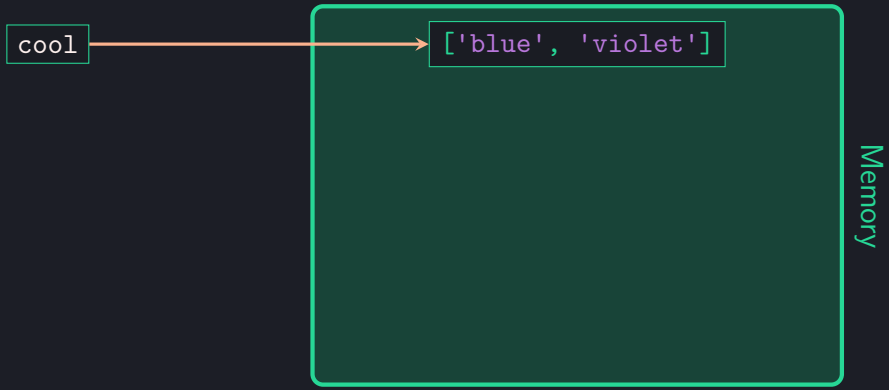
- Presumably, this is allowed with lists (and it is)
- Mutability has some other ramifications though that we want to touch on

Code

Memory

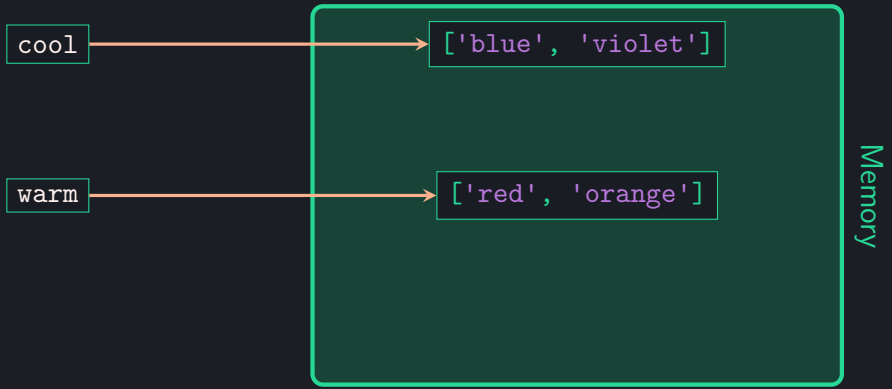
Code

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



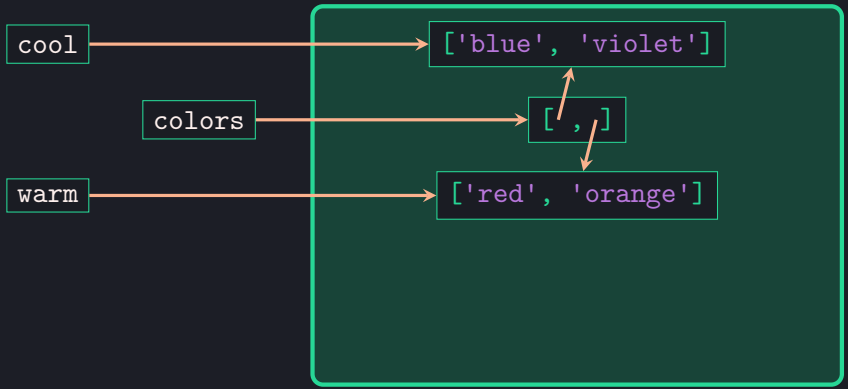
Code

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



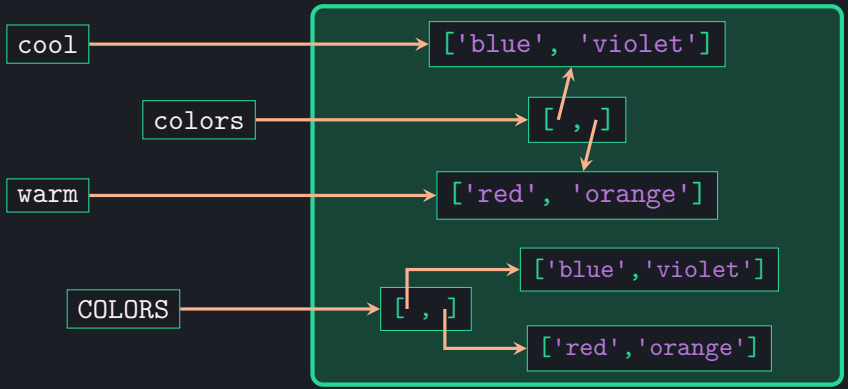
Code

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



Code

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



Code

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