

# LISTS, MUTABILITY

(download slides and .py files to follow along)

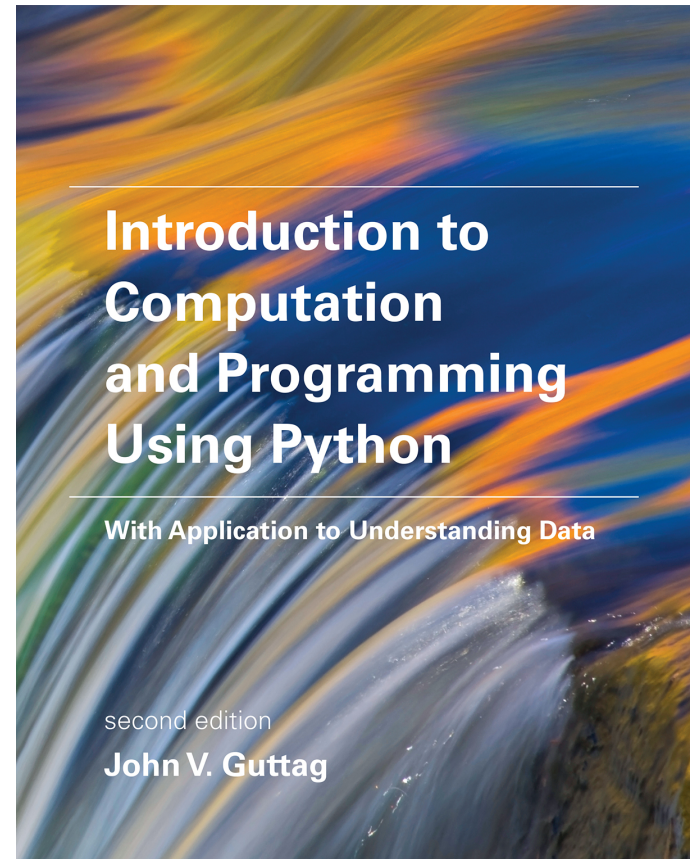
---

6.0001 LECTURE 5

Ana Bell

# ASSIGNED READING

- Sections 5.1 – 5.5
- Sections 4.3 – 4.6



[https://mitpress.mit.edu/sites/default/files/Guttag\\_errata\\_revised\\_083117.pdf](https://mitpress.mit.edu/sites/default/files/Guttag_errata_revised_083117.pdf)

# TODAY

- Have seen variable types: `int`, `float`, `bool`, `string`
- Have introduced new **compound data types**
  - tuples
  - lists
- Today, ideas of
  - Mutability
  - Aliasing
  - Cloning

# TUPLES (RECAP)

- **Indexable ordered sequence** of objects, can mix object types
- Cannot change element values or edit the tuple, **immutable**
- Can index into, slice, concatenate

```
t = ()
```

```
t = (2, "mit", 3)
```

# LISTS (RECAP)

- **Indexable ordered sequence** of objects, can mix object types
- Can index into, slice, concatenate

```
L = []
```

```
L = [2, "mit", 3]
```

- CAN change element values, add items, remove items, essentially edit the list object itself
  - **It's mutable!**

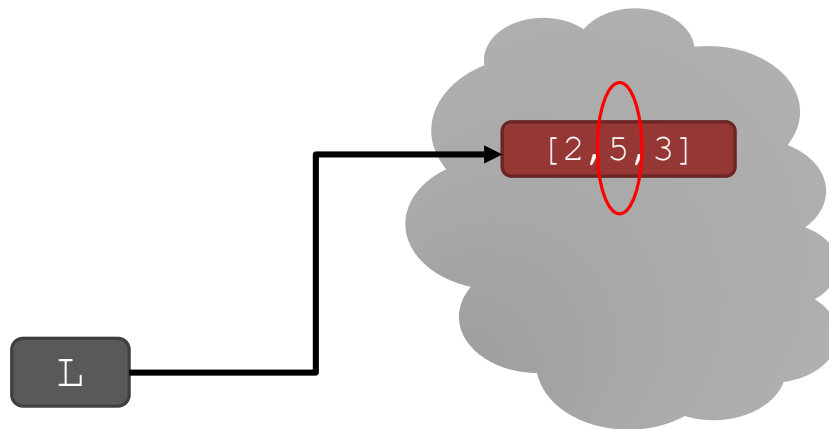
# MUTABILITY (RECAP)

- Lists are **mutable**!
- Assigning to an element at an index **changes** the value

```
L = [2, 1, 3]
```

```
L[1] = 5
```

- L is now [2, 5, 3], note this is the **same object** L



*different from  
strings and tuples!*



# ITERATING OVER A LIST (RECAP)

- Compute the **sum of elements** of a list
- Common pattern

```
total = 0
for i in range(len(L)):
    total += L[i]
print(total)
```

```
total = 0
for i in L:
    total += i
print(total)
```

Like strings, can  
iterate over list  
elements directly

- Notice

- List elements are indexed 0 to  $\text{len}(L) - 1$
- `range(n)` goes from 0 to  $n - 1$

This version is  
more “pythonic”!

# OPERATION ON LISTS: append

- **Add** elements to end of list with `L.append(element)`

- **Mutates** the list!

```
L = [2, 1, 3]
```

```
L.append(5)      → L is now [2, 1, 3, 5]
```



- What is the dot?
  - Lists are Python objects, everything in Python is an object
  - Objects have data
  - Objects have methods and functions
  - Access this information by `object_name.do_something()`
  - Will learn more about these later

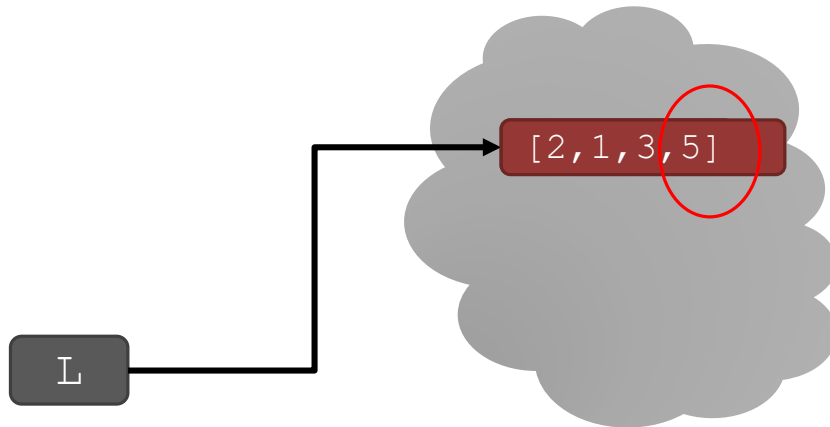


# OPERATION ON LISTS – append

- **Add** element to end of list with `L.append(element)`
- **Mutates** the list!

`L = [2, 1, 3]`

`L.append(5)`      → L is now `[2, 1, 3, 5]`



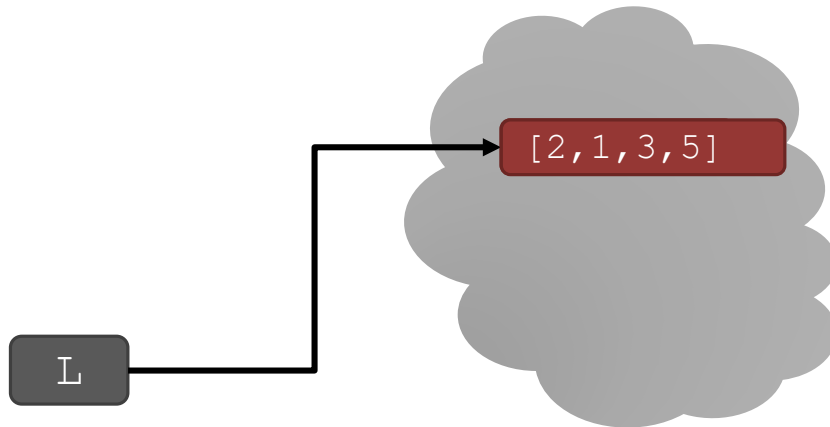
# OPERATION ON LISTS – append

- **Add** element to end of list with `L.append(element)`
- **Mutates** the list!

```
L = [2, 1, 3]
```

```
L.append(5)      → L is now [2, 1, 3, 5]
```

```
L = L.append(5)
```



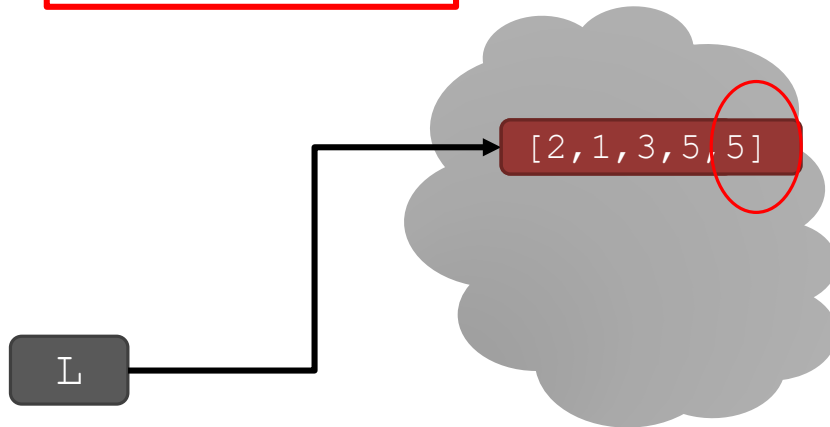
# OPERATION ON LISTS – append

- **Add** element to end of list with `L.append(element)`
- **Mutates** the list!

`L = [2, 1, 3]`

`L.append(5)` → L is now `[2, 1, 3, 5]`

`L = L.append(5)`



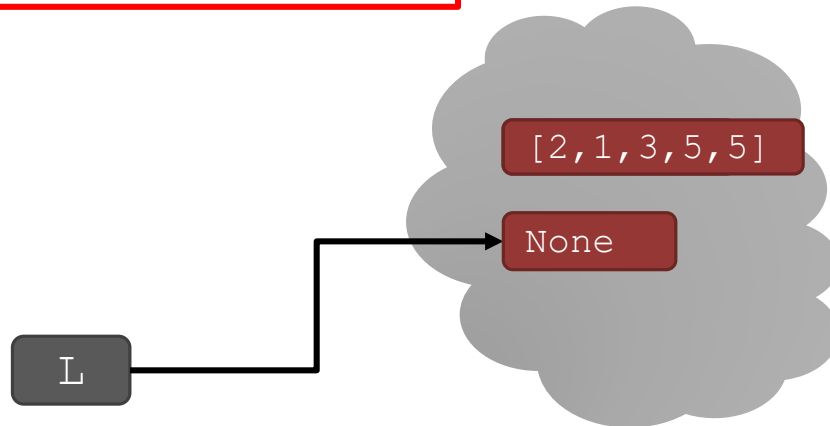
# OPERATION ON LISTS – append

- **Add** element to end of list with `L.append(element)`
- **Mutates** the list!

```
L = [2, 1, 3]
```

```
L.append(5) → L is now [2, 1, 3, 5]
```

```
L = L.append(5)
```



# TRICKY EXAMPLE 1: append

- **Range returns something that behaves like a tuple** (but isn't)
- Generates the first element, and provides an iteration method by which subsequent elements can be generated

`range(5)` → evaluates to tuple `(0, 1, 2, 3, 4)`

`range(2, 6)` → evaluates to tuple `(2, 3, 4, 5)`

```
L = [1, 2, 3, 4]
```

```
for i in range(len(L)):
```

```
    L.append(i)
```

```
print(L)
```

*Iteration sequence is pre-determined at the beginning*

1<sup>st</sup> time: L is [1, 2, 3, 4, 0]

2<sup>nd</sup> time: L is [1, 2, 3, 4, 0, 1]

3<sup>rd</sup> time: L is [1, 2, 3, 4, 0, 1, 2]

4<sup>th</sup> time: L is [1, 2, 3, 4, 0, 1, 2, 3]

# TRICKY EXAMPLE 2: append

```
L = [1, 2, 3, 4]
```

```
i = 0
```

```
for e in L:
```

```
    L.append(i)
```

```
    i += 1
```

```
print(L)
```

*Originally  
[1,2,3,4]*

*L is mutated each iteration*

*1<sup>st</sup> time: L is [1, 2, 3, 4, 0]*

*2<sup>nd</sup> time: L is [1, 2, 3, 4, 0, 1]*

*3<sup>rd</sup> time: L is [1, 2, 3, 4, 0, 1, 2]*

*4<sup>th</sup> time: L is [1, 2, 3, 4, 0, 1, 2, 3]*

*5<sup>th</sup> time: L is [1, 2, 3, 4, 0, 1, 2, 3, 4]*

*...*

# COMBINING LISTS

- **Concatenation**, + operator, creates a **new** list
- **Mutate** list with `L.extend(some_list)`

`L1 = [2, 1, 3]`

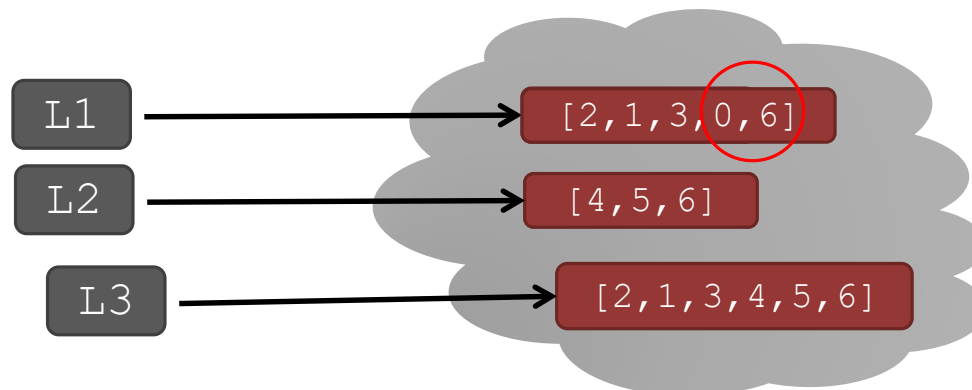
`L2 = [4, 5, 6]`

`L3 = L1 + L2`

→ `L3` is `[2, 1, 3, 4, 5, 6]`

`L1.extend([0, 6])`

→ mutated `L1` to `[2, 1, 3, 0, 6]`



# TRICKY EXAMPLE 3: combining

```
L = [1, 2, 3, 4]  
for e in L:
```

*Originally  
[1,2,3,4]*

```
    L = L + L
```

```
print(L)
```

*L bound to a new object each iteration,  
e in L iterates only 4 times,  
over the original [1,2,3,4]*

1<sup>st</sup> time: new L is [1, 2, 3, 4, 1, 2, 3, 4]

2<sup>nd</sup> time: new L is [1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4]

3<sup>rd</sup> time: new L is [1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4]

4<sup>th</sup> time: new L is [1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4,  
1, 2, 3, 4, 1, 2, 3, 4]



# OPERATION ON LISTS: REMOVE

- Delete element at a **specific index** with `del (L[index])`
- Remove element at **end of list** with `L.pop()`, returns the removed element
- Remove a **specific element** with `L.remove(element)`
  - Looks for the element and removes it
  - If element occurs multiple times, removes first occurrence
  - If element not in list, gives an error


all these  
operations  
mutate  
the list

```
L = [2, 1, 3, 6, 3, 7, 0] # do below in order
L.remove(2) → mutates L = [1, 3, 6, 3, 7, 0]
L.remove(3) → mutates L = [1, 6, 3, 7, 0]
del(L[1])   → mutates L = [1, 3, 7, 0]
L.pop()     → returns 0 and mutates L = [1, 3, 7]
```

# MUTATION AND ITERATION

<http://www.pythontutor.com/> to see step-by-step

- **Avoid** mutating a list as you are iterating over it




```
def remove_dups(L1, L2):  
    for e in L1:  
        if e in L2:  
            L1.remove(e)
```

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

- L1 is [2, 3, 4] not [3, 4] Why?

- Python uses an internal counter to keep track of index it is in the loop
- Mutating changes the list length but Python doesn't update the counter
- Loop never sees element 2



```
def remove_dups(L1, L2):  
    L1_copy = L1[:]  
    for e in L1_copy:  
        if e in L2:  
            L1.remove(e)
```

Clone list first  
Note that `L1_copy = L1`  
does NOT clone



# OTHER LIST OPERATIONS

- `sort()` and `sorted()`

- `reverse()`

- and many more!

<https://docs.python.org/3/tutorial/datastructures.html>

```
L = [9, 6, 0, 3]
```

None  
(watch out  
for this!)

```
a = sorted(L) → returns sorted list, does not mutate L
```

```
a = L.sort() → mutates L = [0, 3, 6, 9]
```

```
L.reverse() → mutates L = [9, 6, 3, 0]
```

# MUTATION, ALIASING, CLONING



IMPORTANT  
and  
TRICKY!

***Again, Python Tutor is your best friend  
to help sort this out!***

<http://www.pythontutor.com/>

# LISTS IN MEMORY

- Lists are **mutable**
- Behave differently than immutable types
- Is an object in memory
- Variable name points to object
- Using equal sign between mutable objects creates aliases
- Any variable pointing to that object is affected
- Key phrase to keep in mind when working with lists is **side effects**

# ALIASING



Boston  
The Hub  
Beantown

- City may be known by many names
- Attributes of a city
  - small, tech-savvy
- All nicknames point to the **same city**
  - add new attribute to **one nickname** ...

Boston

small

tech-savvy

snowy

... all the **aliases** refer to the old attribute and all the new ones

The Hub

small

tech-savvy

snowy

Beantown

small

tech-savvy

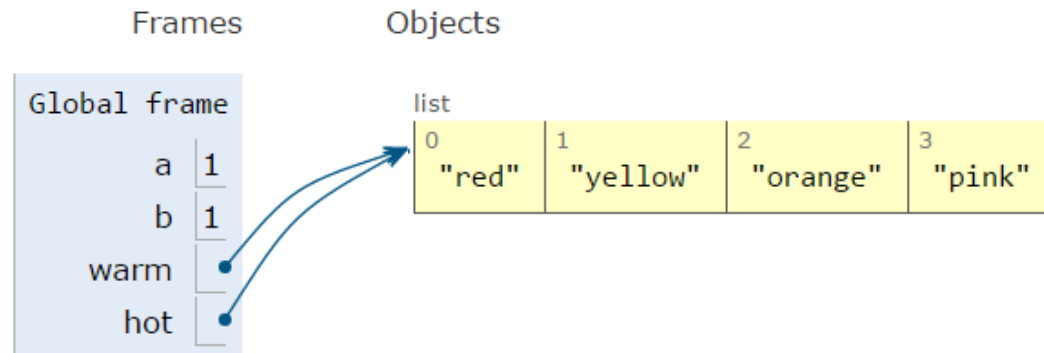
snowy

# ALIASES

- `hot` is an **alias** for `warm` – changing one changes the other!
- `append()` has a side effect

```
1 a = 1
2 b = a
3 print(a)
4 print(b)
5
6 warm = ['red', 'yellow', 'orange']
7 hot = warm
8 hot.append('pink')
9 print(hot)
10 print(warm)
```

```
1
1
['red', 'yellow', 'orange', 'pink']
['red', 'yellow', 'orange', 'pink']
```

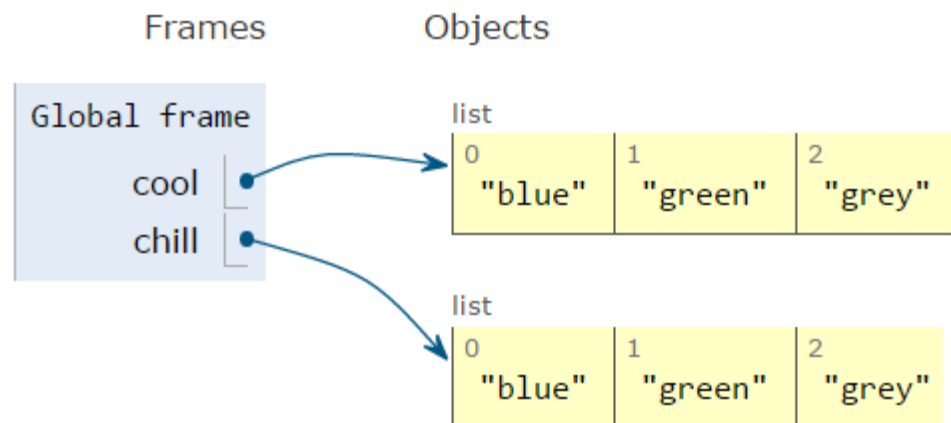


# CLONING A LIST

- Create a new list and **copy every element** using  
`chill = cool[:]`

```
1 cool = ['blue', 'green', 'grey']  
2 chill = cool[:]  
3 chill.append('black')  
4 print(chill)  
5 print(cool)
```

```
['blue', 'green', 'grey', 'black']  
['blue', 'green', 'grey']
```



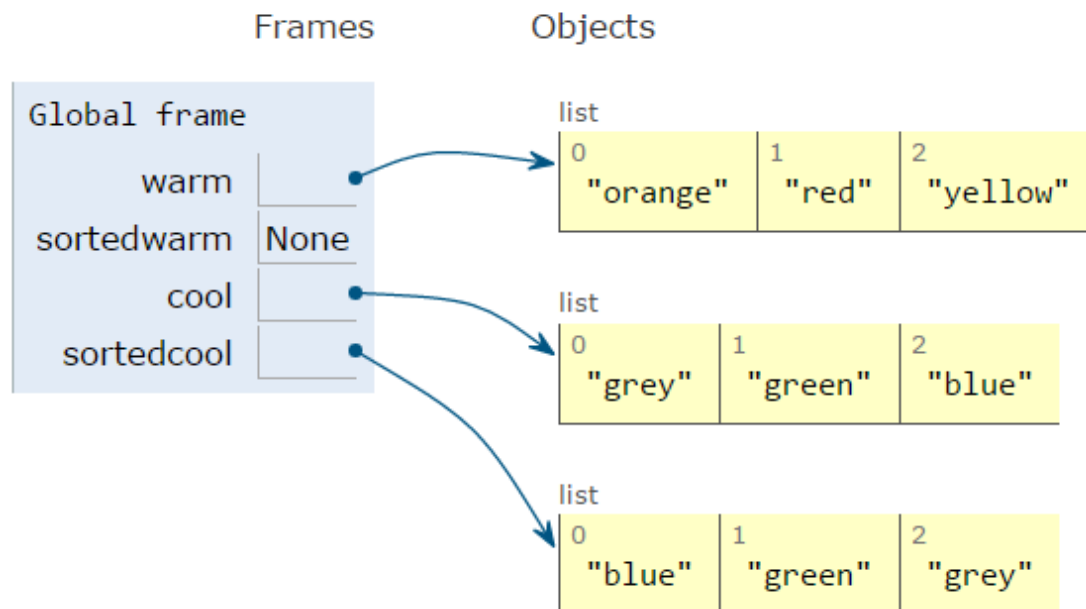


# SORTING LISTS

- Calling `sort()` **mutates** the list, returns nothing
- Calling `sorted()` **does not mutate** list, must assign result to a variable

```
['orange', 'red', 'yellow']  
None  
['grey', 'green', 'blue']  
['blue', 'green', 'grey']
```

```
1 warm = ['red', 'yellow', 'orange']  
2 sortedwarm = warm.sort()  
3 print(warm)  
4 print(sortedwarm)  
5  
6 cool = ['grey', 'green', 'blue']  
7 sortedcool = sorted(cool)  
8 print(cool)  
9 print(sortedcool)
```



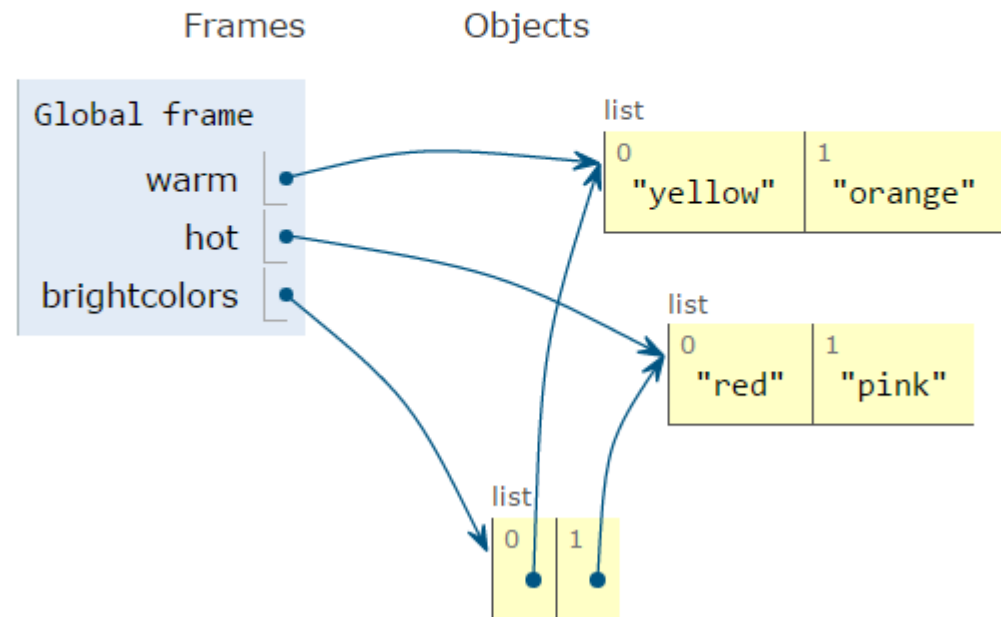


# LISTS OF LISTS OF LISTS OF....

- Can have **nested** lists
- Side effects still possible after mutation

```
[['yellow', 'orange'], ['red']]  
['red', 'pink']  
[['yellow', 'orange'], ['red', 'pink']]
```

```
1 warm = ['yellow', 'orange']  
2 hot = ['red']  
3 brightcolors = [warm]  
4 brightcolors.append(hot)  
5 print(brightcolors)  
6 hot.append('pink')  
7 print(hot)  
8 print(brightcolors)
```



# Monday

- Dictionaries
- Exceptions
- Assertions
- Debugging

# 5 Min Break, then Quiz Time!

- Sit at a seat, **not on the floor**
- No aids allowed, **only MITx and your IDE**
- If you finish early, **stay in your seat** (no phones, external websites, etc)
- **Checkout password given in the last 2 mins of exam**
- **Exam link:**