TUPLES, LISTS, ALIASING, MUTABILITY, CLONING

TUPLES

- an ordered sequence of elements, can mix element types
- cannot change element values, **immutable**
- represented with parentheses

remember strings?

`te = ()` empty tuple

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

`t[0]` → evaluates to 2

`(2,"mit",3) + (5,6)` → evaluates to `(2,"mit",3,5,6)`

`t[1:2]` → slice tuple, evaluates to `("mit",)`

`t[1:3]` → slice tuple, evaluates to `("mit",3)`

extra comma means a tuple with one element

`len(t)` → evaluates to 3

`t[1] = 4` → gives error, can't modify object

TUPLES

- conveniently used to **swap** variable values

```
x = y
y = x
```

✗

```
temp = x
x = y
y = temp
```

✓

```
(x, y) = (y, x)
```

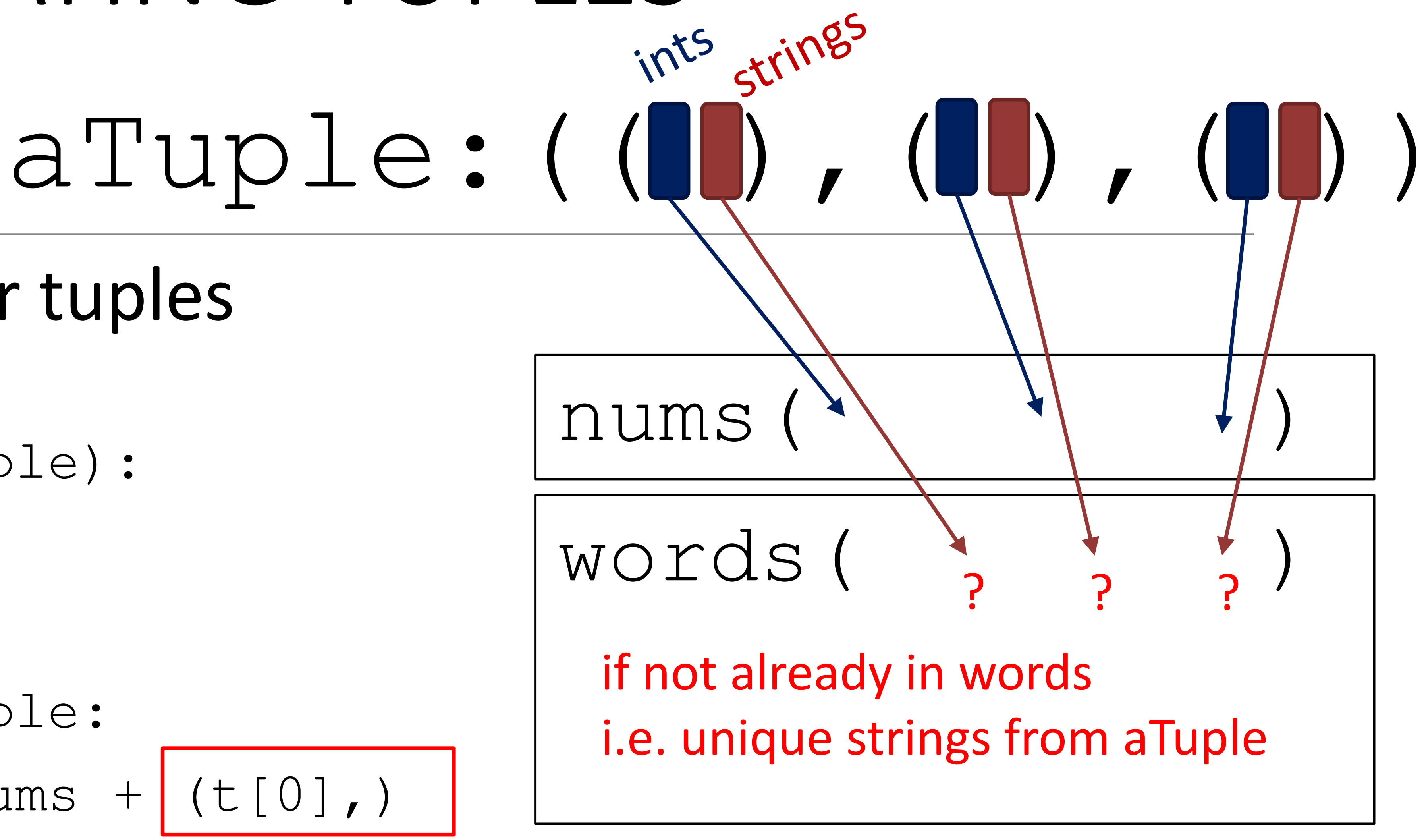
✓

- used to **return more than one value** from a function

```
def quotient_and_remainder(x, y):
    q = x // y
    r = x % y
    return (q, r)
(quot, rem) = quotient_and_remainder(4,5)
```

integer division

MANIPULATING TUPLES



if not already in words
i.e. unique strings from aTuple

- can **iterate** over tuples

```
def get_data(aTuple):
    nums = ()
    words = ()
    for t in aTuple:
        nums = nums + (t[0],)
        if t[1] not in words:
            words = words + (t[1],)
    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)
```

empty tuple

singleton tuple

LISTS

- **ordered sequence** of information, accessible by index
- a list is denoted by **square brackets**, `[ ]`
- a list contains **elements**
  - usually homogeneous (ie, all integers)
  - can contain mixed types (not common)
- list elements can be changed so a list is **mutable**

INDICES AND ORDERING

`a_list = []` empty list

`L = [2, 'a', 4, [1,2]]`

`len(L)` → evaluates to 4

`L[0]` → evaluates to 2

`L[2]+1` → evaluates to 5

`L[3]` → evaluates to `[1,2]`, another list!

`L[4]` → gives an error

`i = 2`

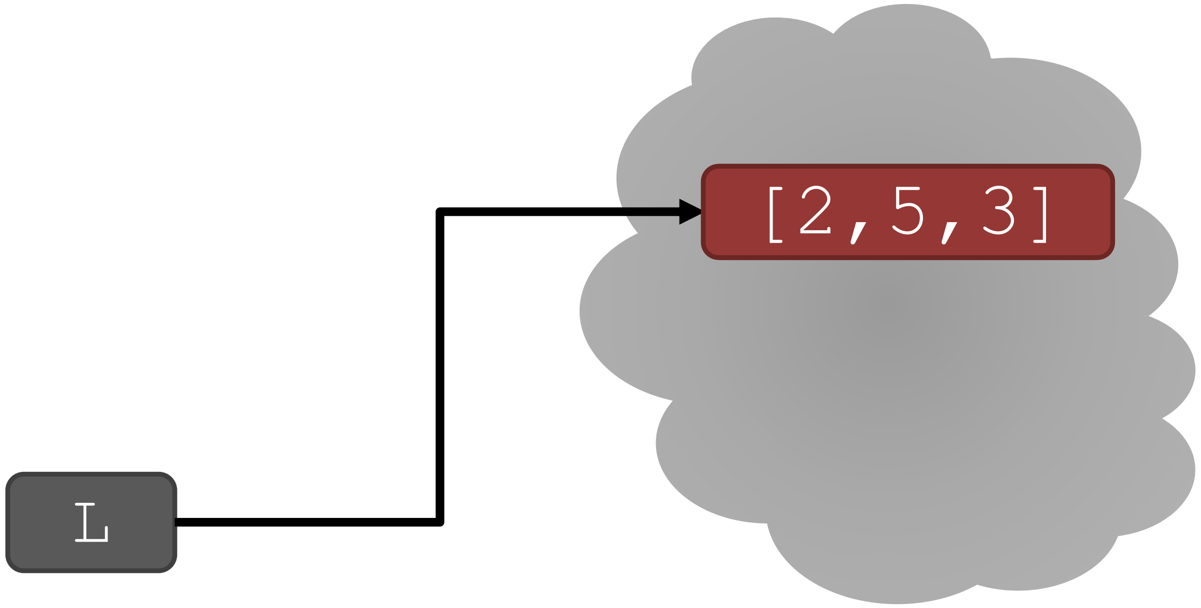
`L[i-1]` → evaluates to 'a' since `L[1]='a'` above

CHANGING ELEMENTS

- 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`



ITERATING OVER A LIST

- compute the **sum of elements** of a list
- common pattern, iterate over list elements

```
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 `len(L)-1`
  - `range(n)` goes from `0` to `n-1`

OPERATIONS ON LISTS - ADD

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

modifies 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

OPERATIONS ON LISTS - ADD

- to combine lists together use **concatenation**, + operator, to give you a new list
- **modify** 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, L2 unchanged

L1.extend([0,6])          → modified L1 to [2,1,3,0,6]
```

OPERATIONS 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) → modifies L = [1,3,6,3,7,0]
L.remove(3) → modifies L = [1,6,3,7,0]
del(L[1])   → modifies L = [1,3,7,0]
L.pop()     → returns 0 and modifies L = [1,3,7]
```

CONVERT LISTS TO STRINGS AND BACK

- convert **string to list** with `list(s)`, returns a list with every character from `s` an element in `L`
- can use `s.split()`, to **split a string on a character** parameter, splits on spaces if called without a parameter
- use `''.join(L)` to turn a **list of characters into a string**, can give a character in quotes to add char between every element

```
s = "I<3 cs"            →  s is a string
list(s)                 → returns ['I','<','3',' ','c','s']
s.split('<')            → returns ['I', '3 cs']
L = ['a','b','c']       →  L is a list
''.join(L)              → returns "abc"
'_'.join(L)             → returns "a_b_c"
```

OTHER LIST OPERATIONS

- `sort()` and `sorted()`
- `reverse()`
- and many more!

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

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

`sorted(L)` → returns sorted list, does **not** modify `L`

`L.sort()` → modifies `L=[0,3,6,9]`

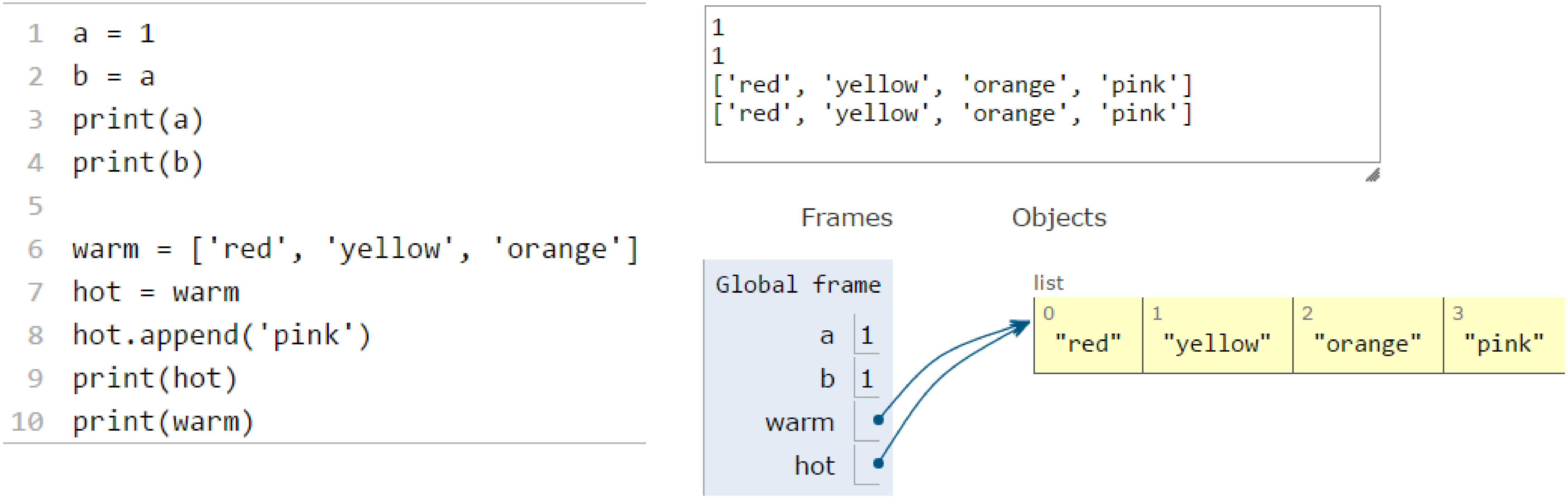
`L.reverse()` → modifies `L=[9,6,3,0]`

LISTS IN MEMORY

- lists are **mutable**
- behave differently than immutable types
- is an object in memory
- variable name points to object
- any variable pointing to that object is affected
- key phrase to keep in mind when working with lists is **side effects**

ALIASES

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



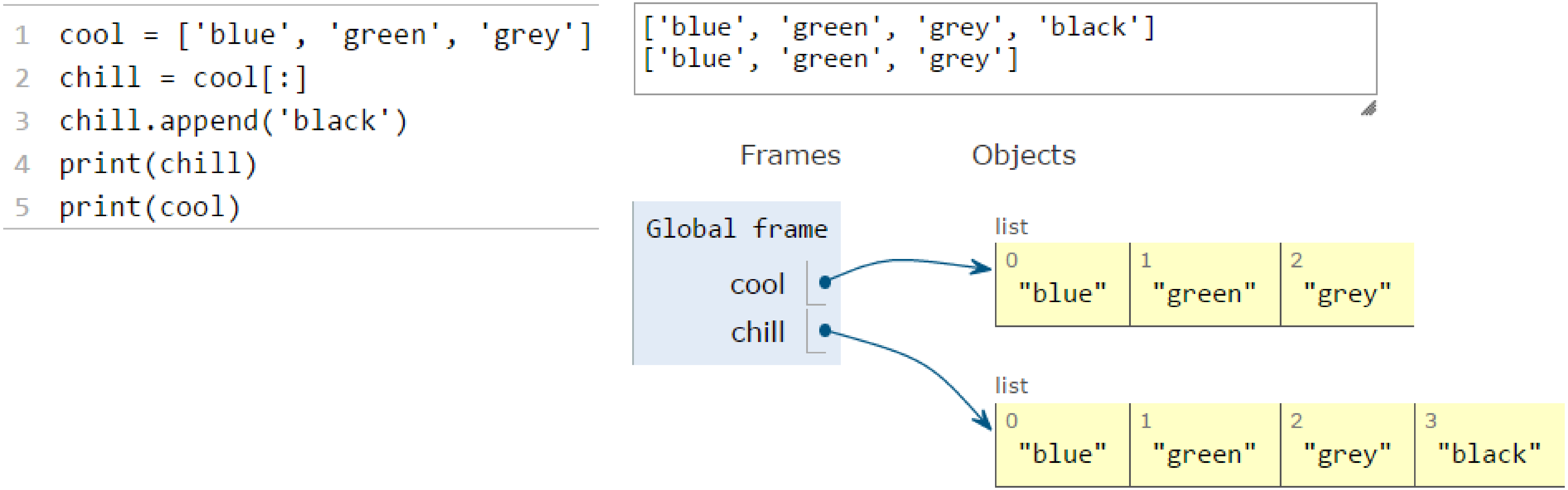
```
a = 1
b = a
print(a)
print(b)

warm = ['red', 'yellow', 'orange']
hot = warm
hot.append('pink')
print(hot)
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[:]`

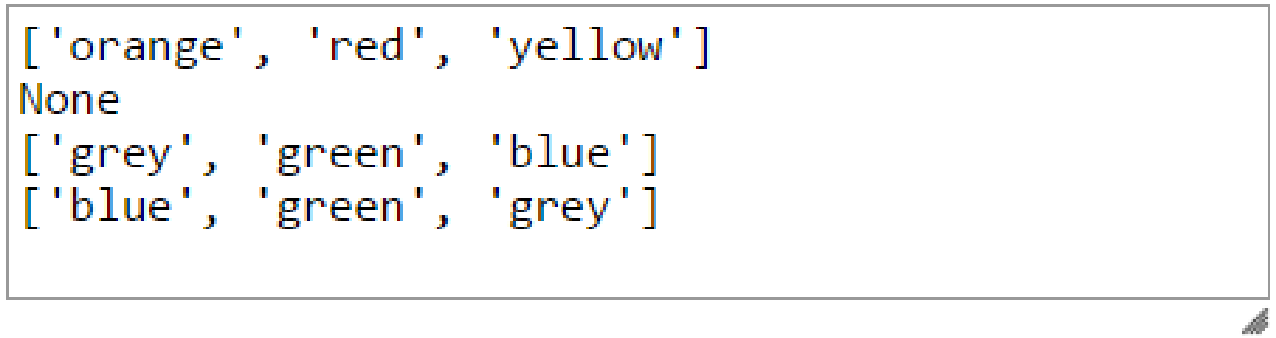


```python
cool = ['blue', 'green', 'grey']
chill = cool[:]
chill.append('black')
print(chill)
print(cool)
```

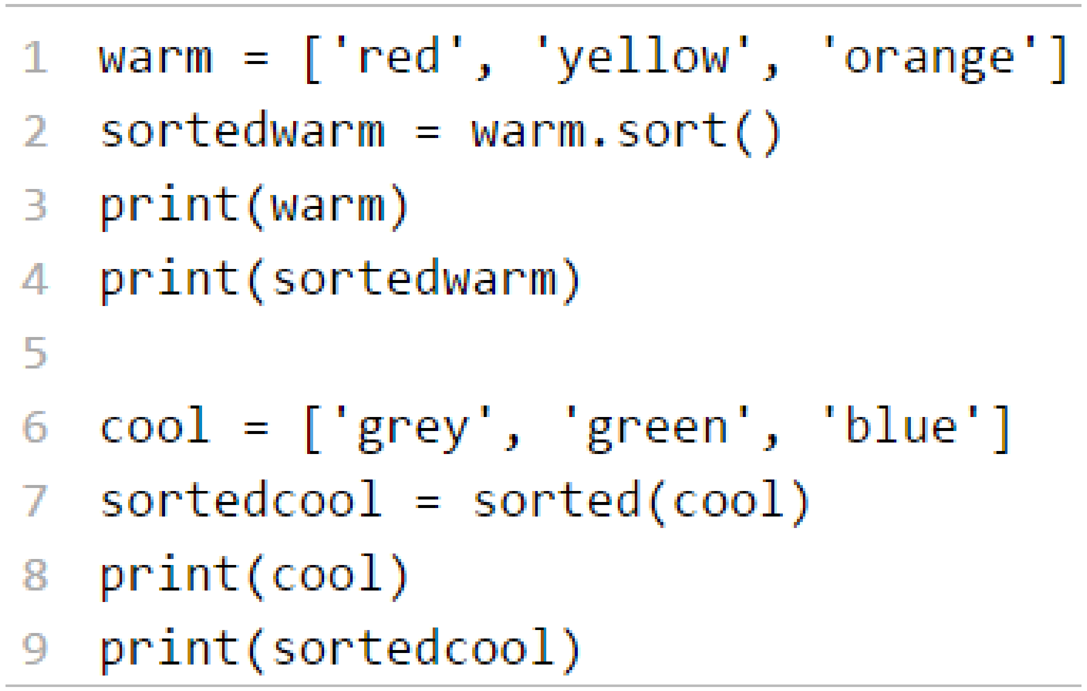
```
['blue', 'green', 'grey', 'black']
['blue', 'green', 'grey']
```

SORTING LISTS

- calling `sort()` modifies the list, returns nothing
- calling `sorted()` **does not** modify list, must assign result to a variable

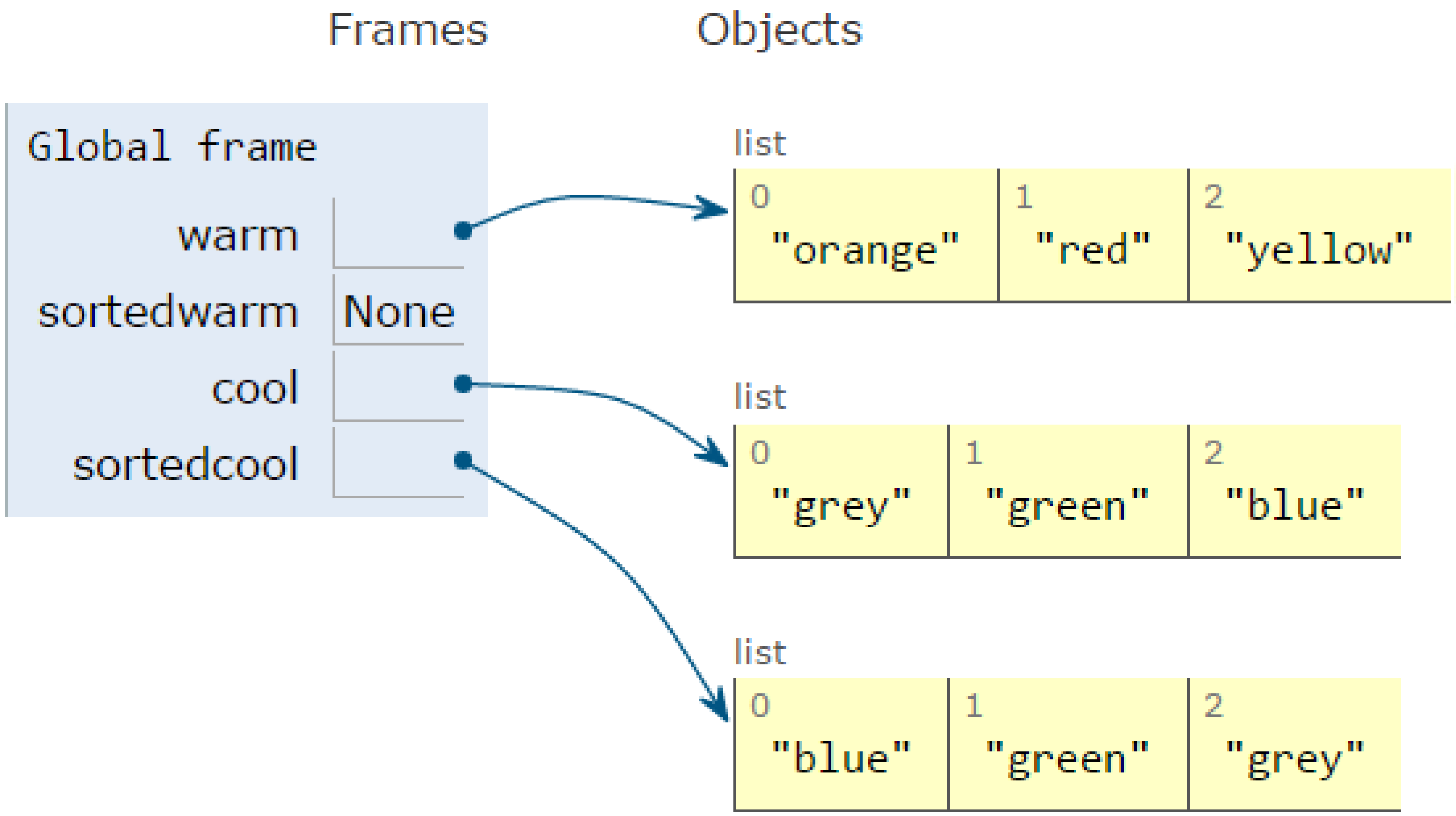


```
['orange', 'red', 'yellow']
None
['grey', 'green', 'blue']
['blue', 'green', 'grey']
```



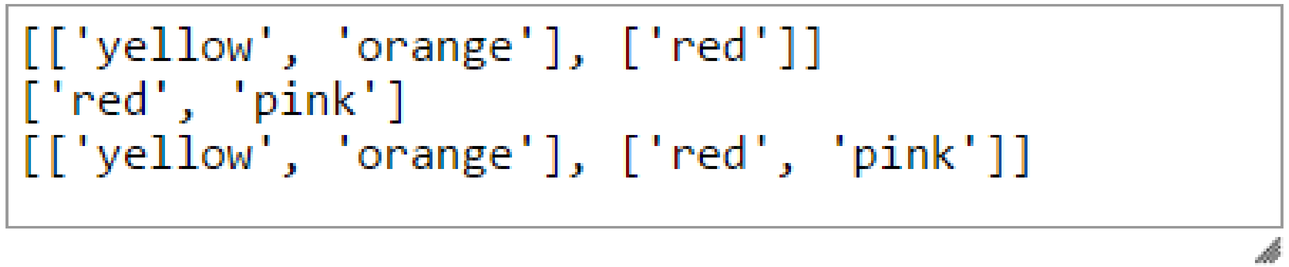
```
warm = ['red', 'yellow', 'orange']
sortedwarm = warm.sort()
print(warm)
print(sortedwarm)

cool = ['grey', 'green', 'blue']
sortedcool = sorted(cool)
print(cool)
print(sortedcool)
```

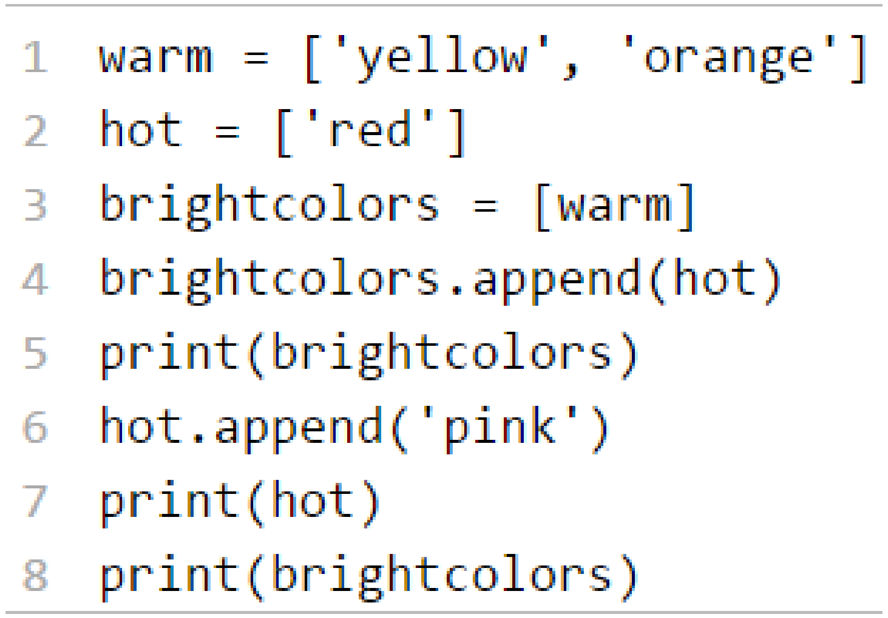


LISTS OF LISTS OF LISTS OF....

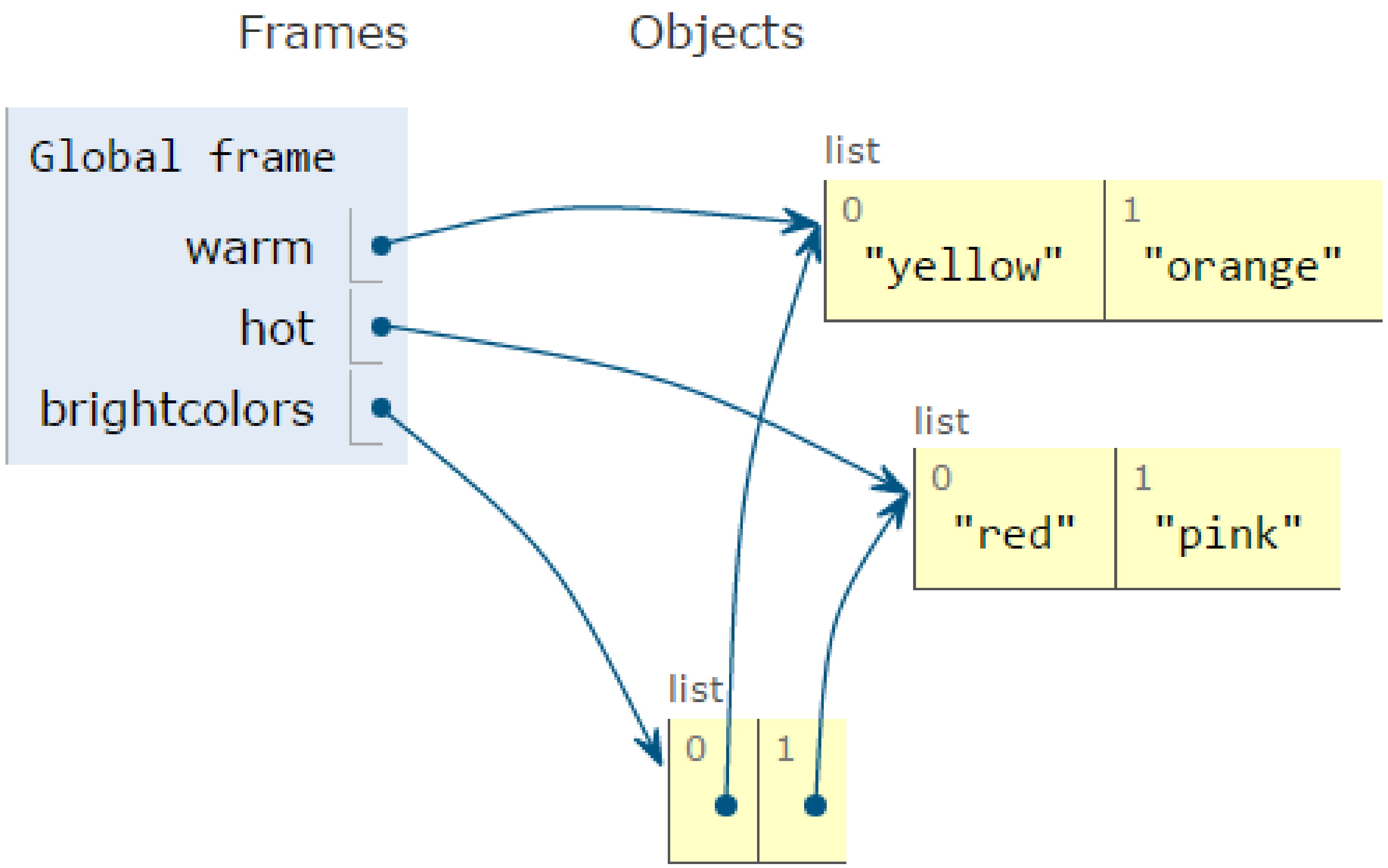
- can have **nested** lists
- side effects still possible after mutation



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



```
warm = ['yellow', 'orange']
hot = ['red']
brightcolors = [warm]
brightcolors.append(hot)
print(brightcolors)
hot.append('pink')
print(hot)
print(brightcolors)
```



MODIFYING AND ITERATION
Try this in Python Tutor!

- **avoid** changing list as you are iterating over it

✗

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

✓

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

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

clone list first, note that `L1_copy = L1` does NOT clone

- `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