LECTURE 10: LISTS

Hung-Yu Wei

List

```
[10, 20, 30, 40]
['crunchy frog', 'ram bladder', 'lark vomit']
```

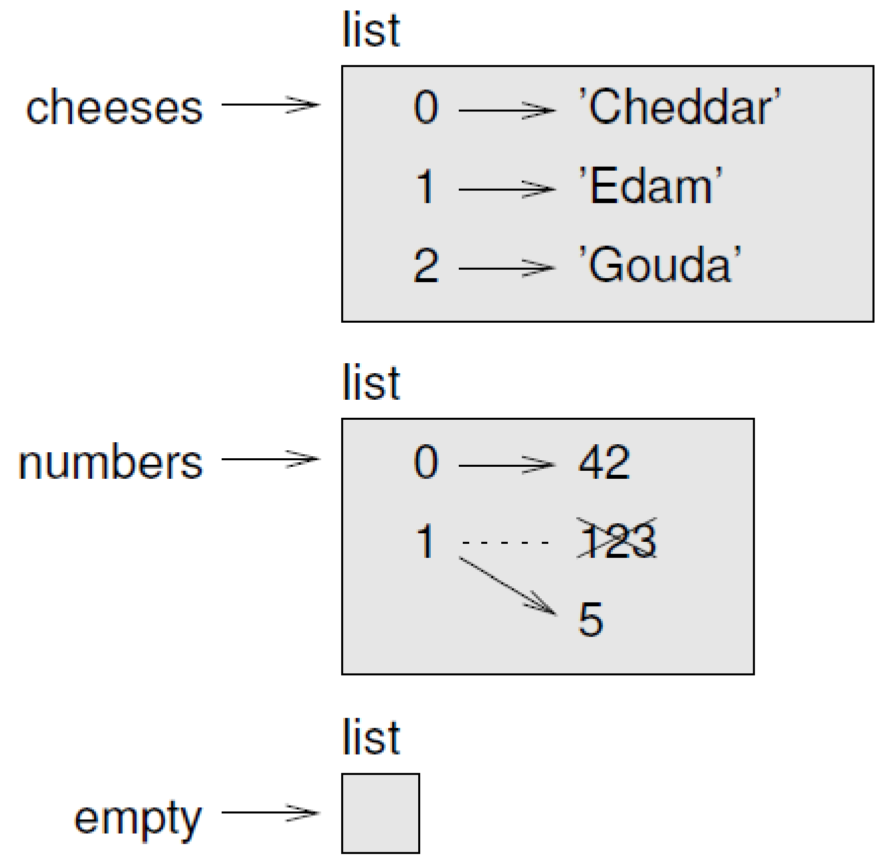
- List
  - *A sequence*
  - *Any type*
  - *Elements (items)*

```
['spam', 2.0, 5, [10, 20]]
```

- Nested list
- Empty list

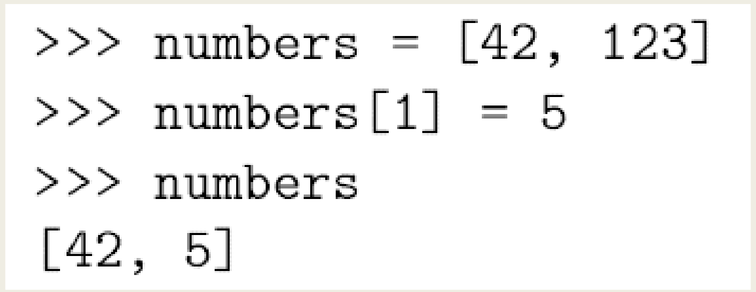
```
empty = []
```

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [42, 123]
>>> empty = []
>>> print(cheeses, numbers, empty)
['Cheddar', 'Edam', 'Gouda'] [42, 123] []
```

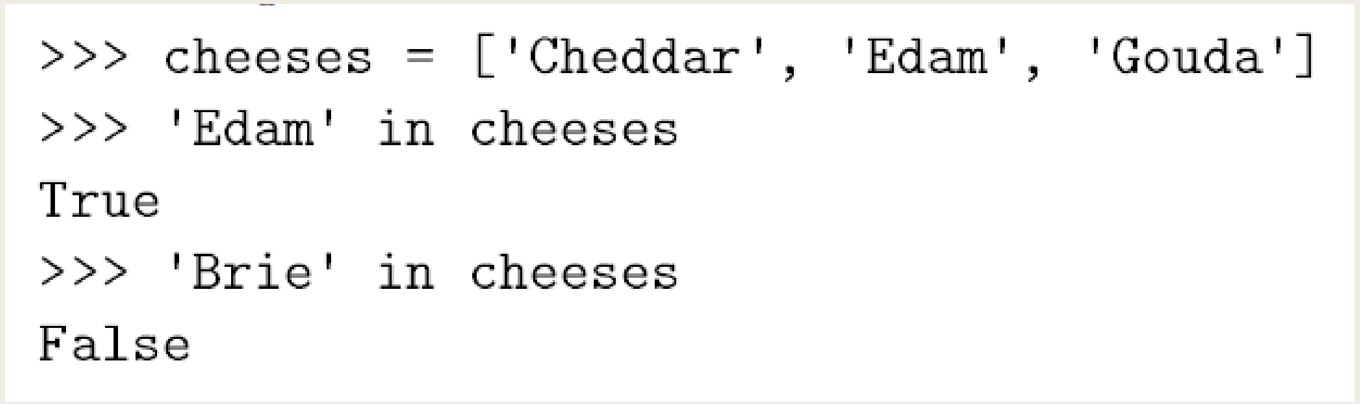


Access element in a list

- List is mutable (can modify element)
- index starting from 0
- Any integer expression can be used as an index.
  - *read/write an element that does not exist ➔ IndexError.*
- index has a negative value
  - *count backward from the end of the list.*

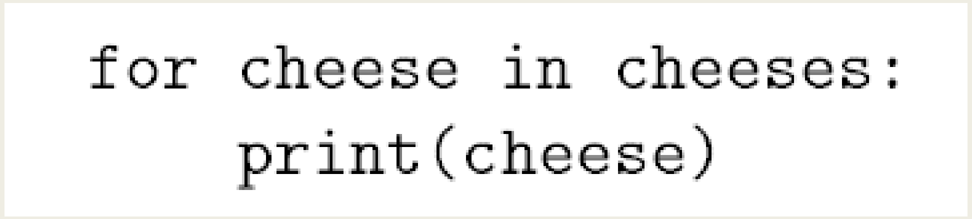


```
>>> numbers = [42, 123]
>>> numbers[1] = 5
>>> numbers
[42, 5]
```

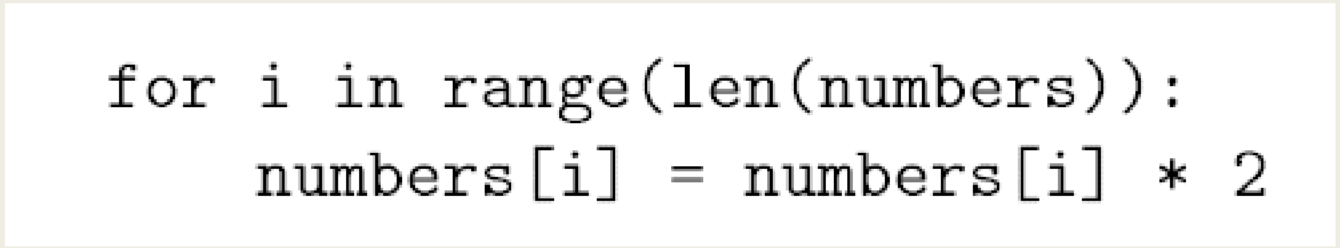


```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

Traversing a list



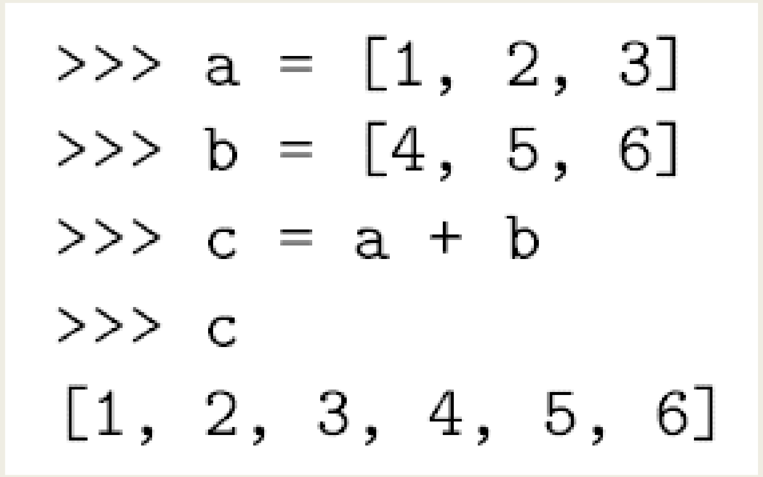
```
for cheese in cheeses:
    print(cheese)
```



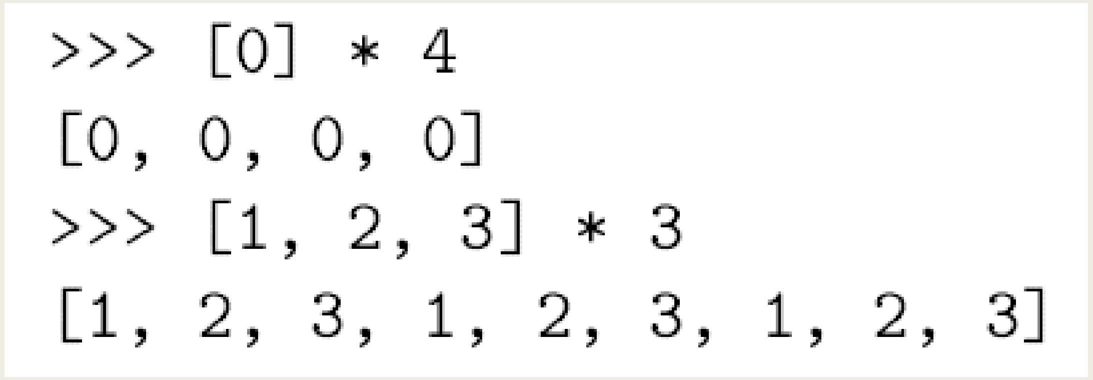
```
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

List operations

- +
  - *Conconcatenation*
- *
  - *Repeat n times*



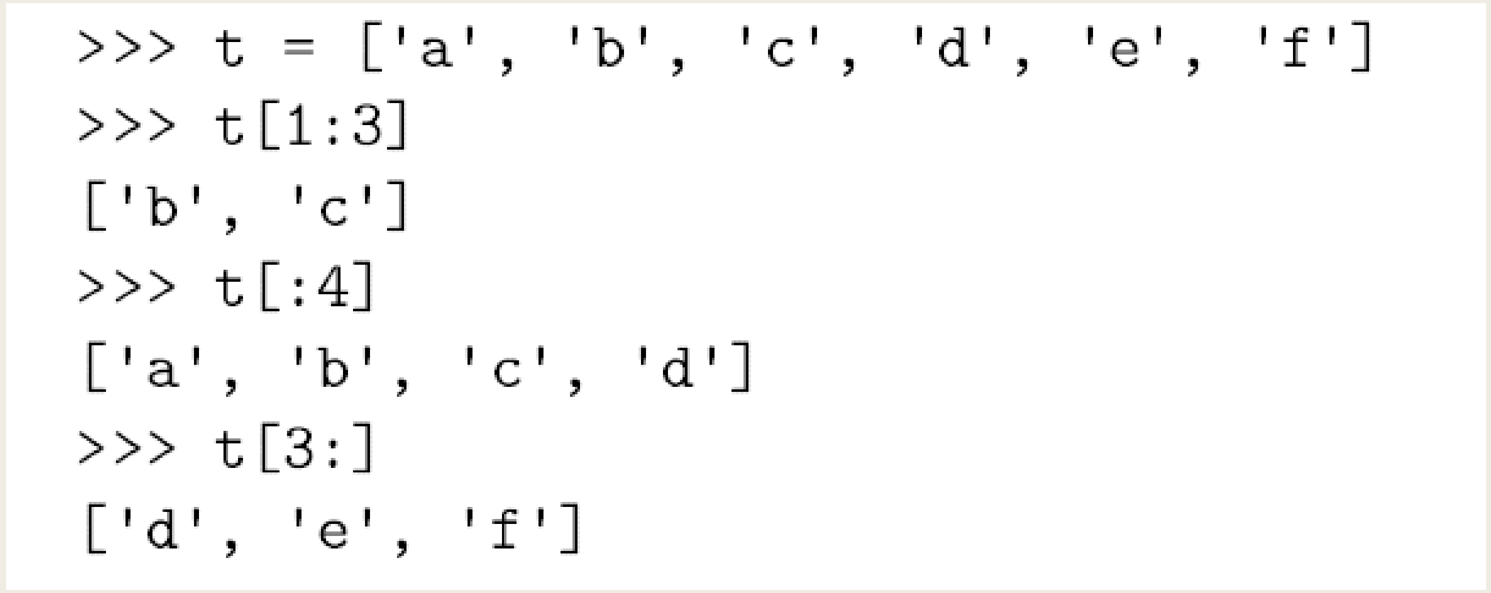
```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
```



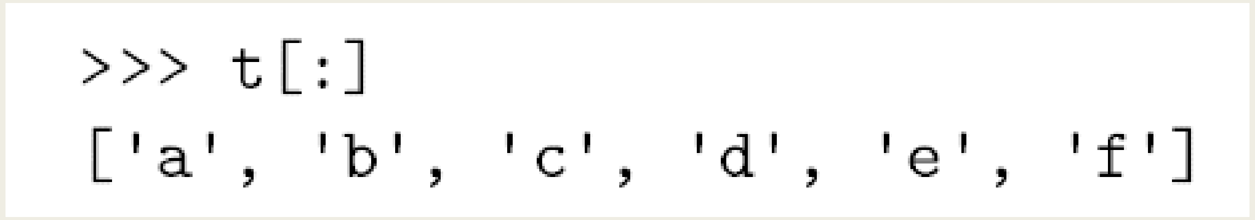
```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Slices in list

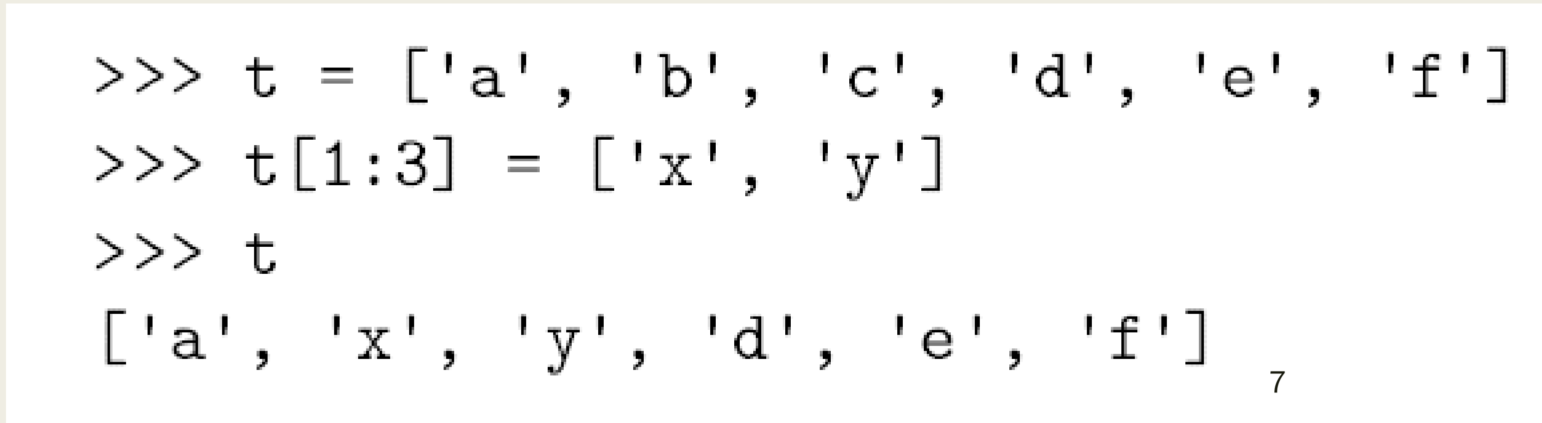
- [n:m]
- [:]
  - *Whole list*
- [n:]
- [:m]



```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']
```



```
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```



```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> t
['a', 'x', 'y', 'd', 'e', 'f']
```

List methods

- Append (an element)
- Extend (a list)
- Sort

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd']
```

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
```

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

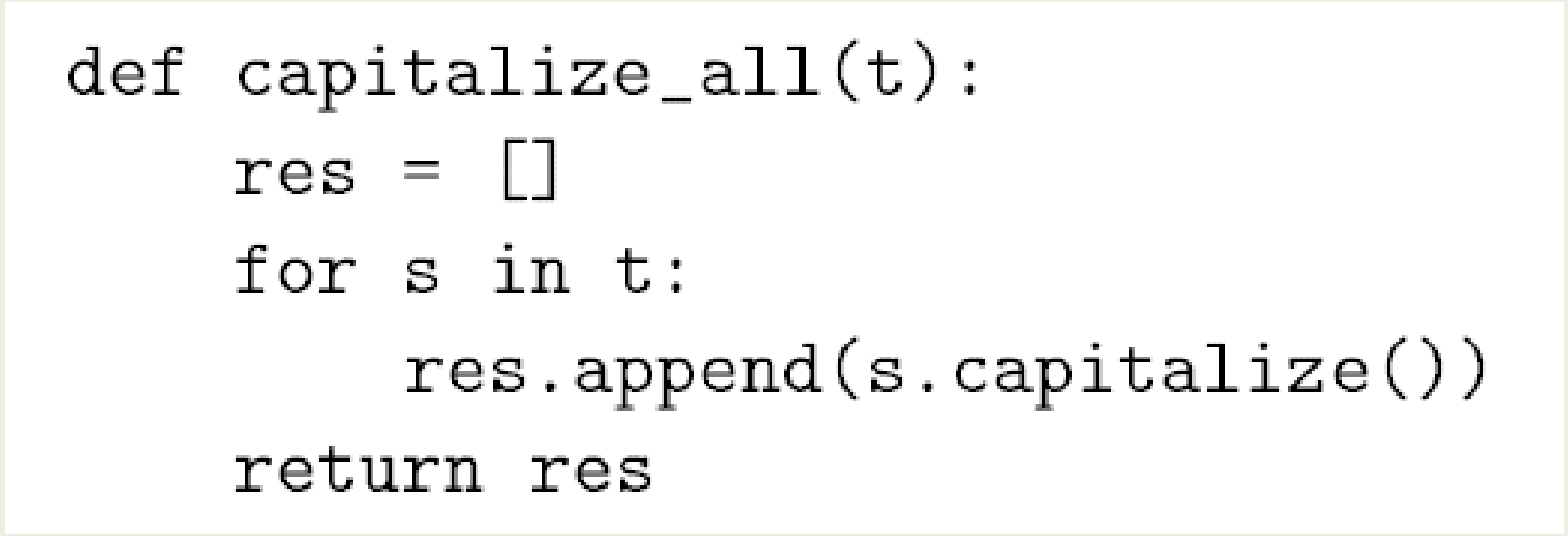
Example: add all elements in list

```
def add_all(t):
    total = 0
    for x in t:
        total += x
    return total
```

total **+=** x
total = total + x

```
>>> t = [1, 2, 3]
>>> sum(t)
6
```

Example: capitalize list

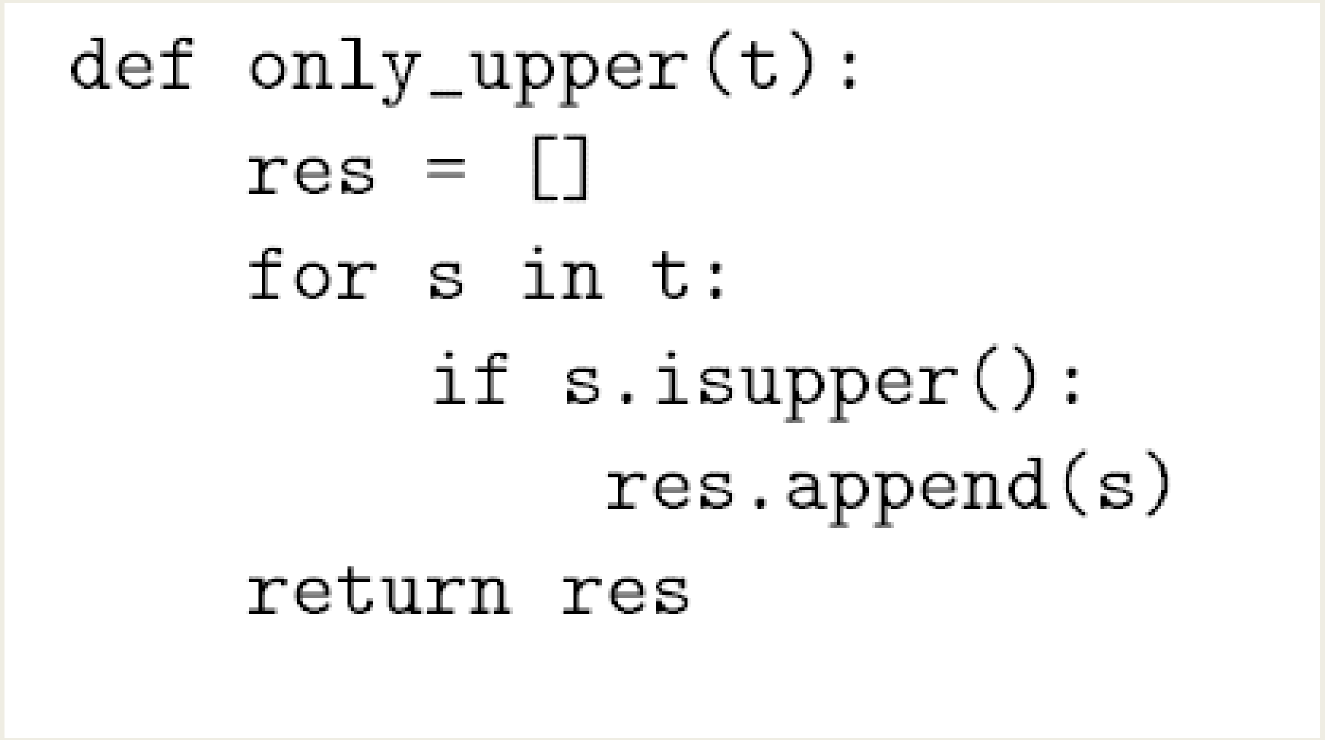


```python
def capitalize_all(t):
    res = []
    for s in t:
        res.append(s.capitalize())
    return res
```

Map, filter, and reduce

- Map
  - *It "maps" a function onto each of the elements in a sequence*
  - *E.g. capitalize*
- Filter
  - *E.g. only get strings with upper case letters*
- Reduce
  - *Summarize*
  - *E.g. summation of all the values in the list*
- MapReduce
  - *https://en.wikipedia.org/wiki/MapReduce*
  - *A MapReduce program is composed of a* ***map*** *procedure (or method), which performs* ***filtering*** *and sorting (such as sorting students by first name into queues, one queue for each name), and a* ***reduce*** *method, which performs a* ***summary*** *operation (such as counting the number of students in each queue, yielding name frequencies).*

Example: select the upper-case elements

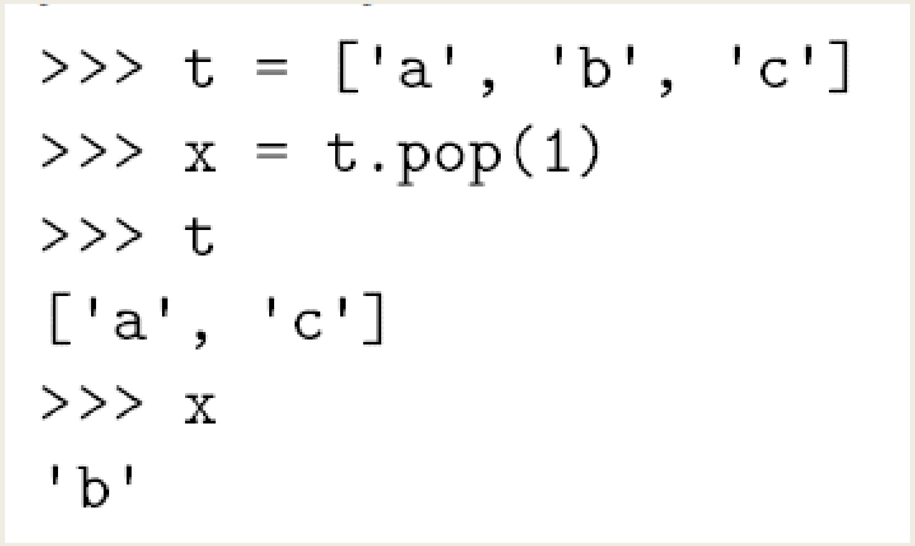


```python
def only_upper(t):
    res = []
    for s in t:
        if s.isupper():
            res.append(s)
    return res
```

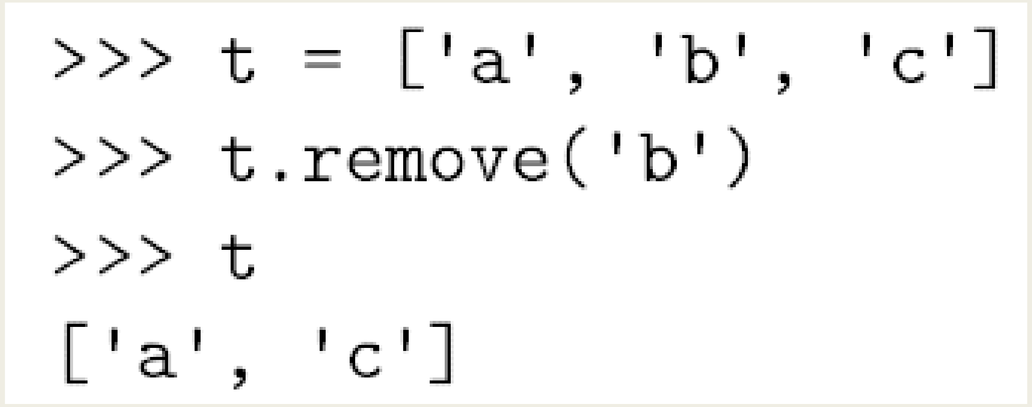
- myString.isupper()
  - *Return True if the string only contain upper case letter*

Deleting elements (1)

- pop
  - *t.pop(index)*
  - *You can get the removed value*
- Remove
  - *t.remove('element_to_remove')*
  - *You know the element that you want to remove*
- Delete
  - *del t[index]*
  - *You don't need the removed value*



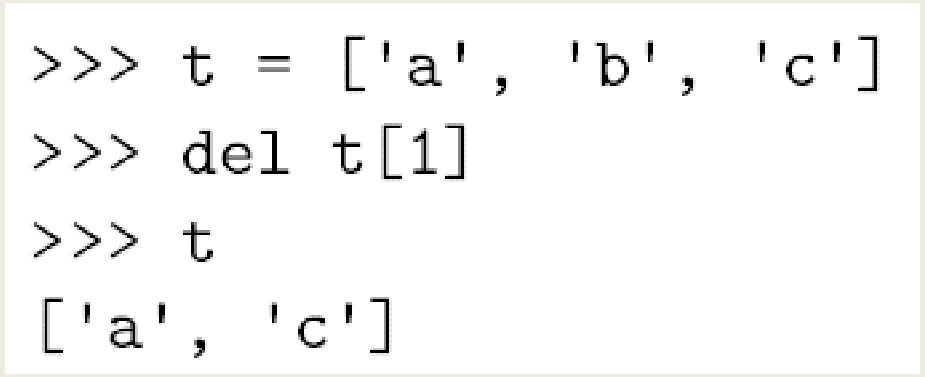
```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> t
['a', 'c']
>>> x
'b'
```



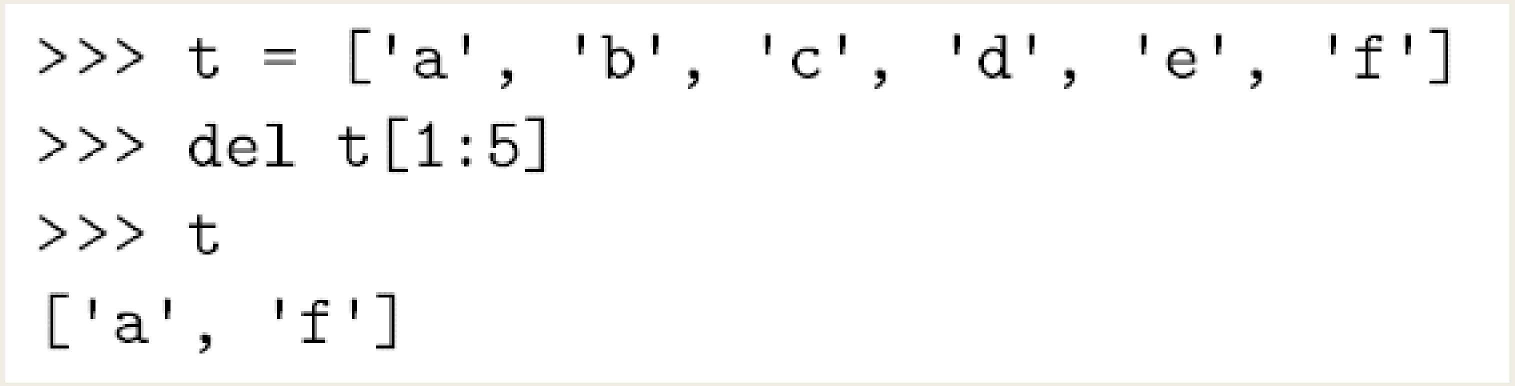
```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> t
['a', 'c']
```

Deleting elements (2)

- Delete
  - *del t[index]*



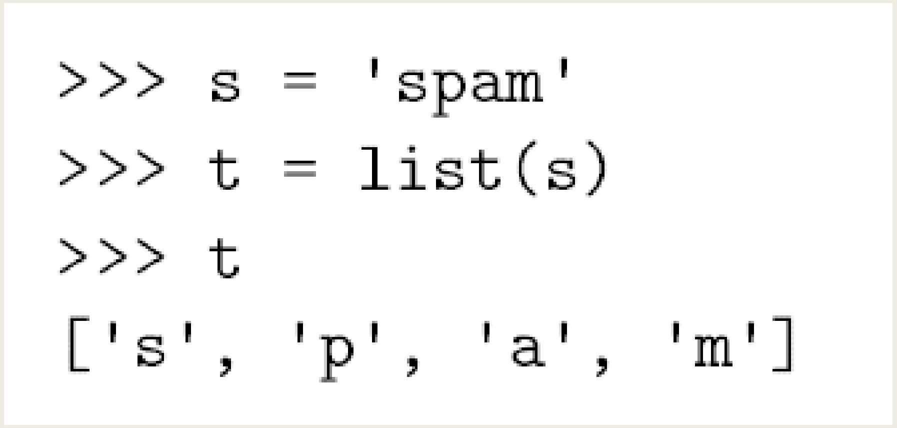
```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> t
['a', 'c']
```



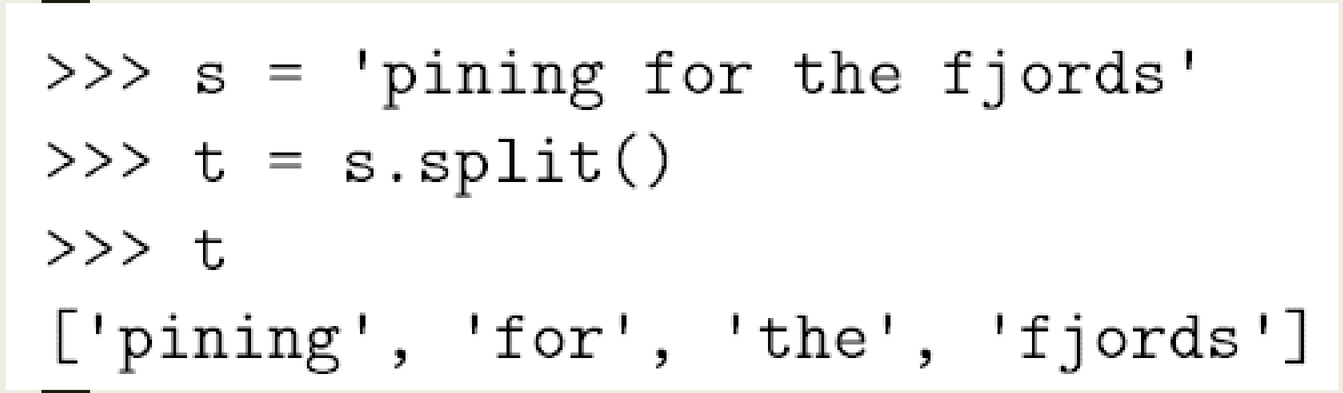
```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> t
['a', 'f']
```

List and strings (1)

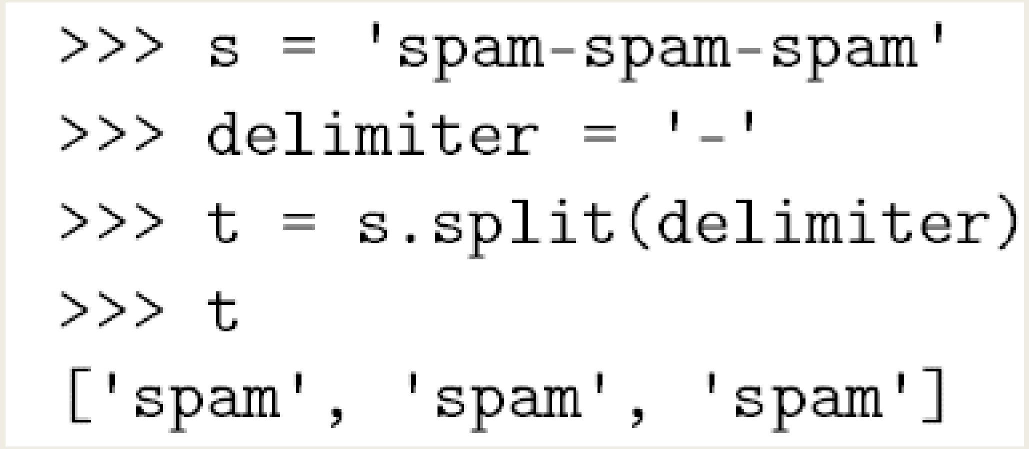
- Convert a string to list
  - *list(s)*
- Break a string into words
  - *s.split()*
  - *s.split(delimiter)*



```
>>> s = 'spam'
>>> t = list(s)
>>> t
['s', 'p', 'a', 'm']
```



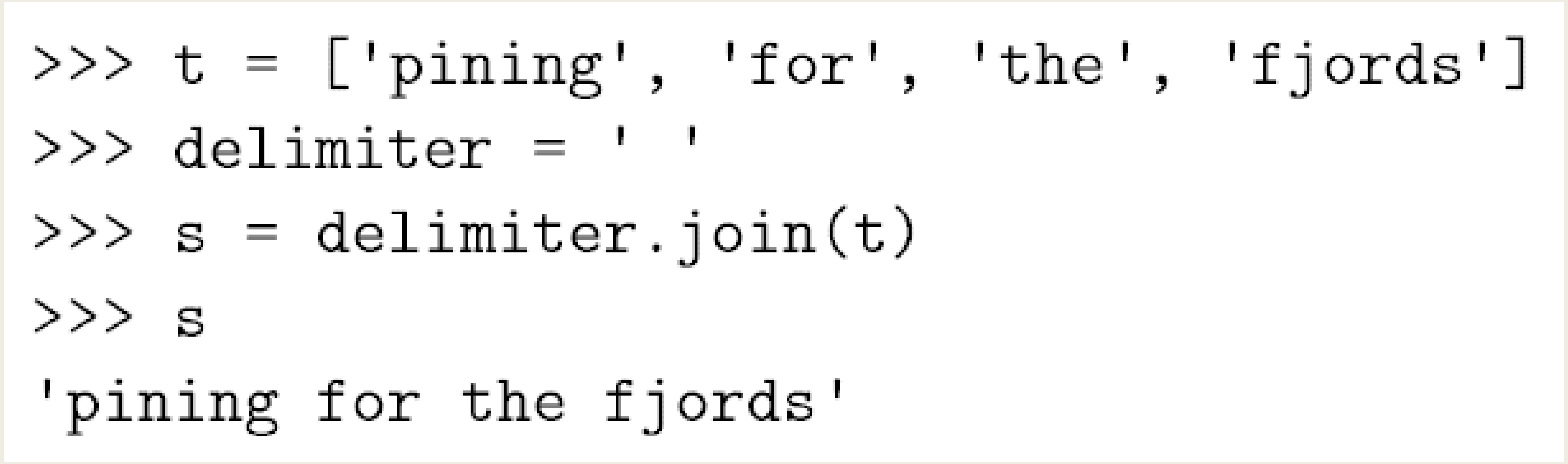
```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> t
['pining', 'for', 'the', 'fjords']
```



```
>>> s = 'spam-spam-spam'
>>> delimiter = '-'
>>> t = s.split(delimiter)
>>> t
['spam', 'spam', 'spam']
```

List and strings (2)

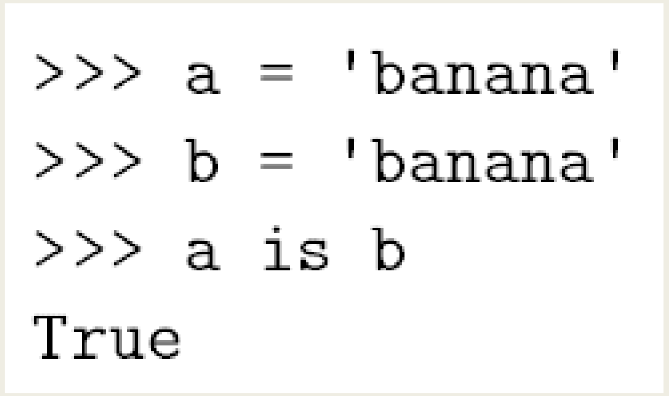
- join
  - *Concatenate a list (of strings) into a string*



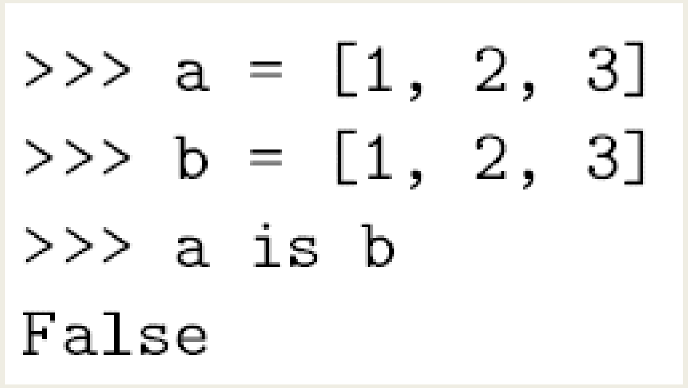
```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> s = delimiter.join(t)
>>> s
'pining for the fjords'
```

equivalent v.s. identical

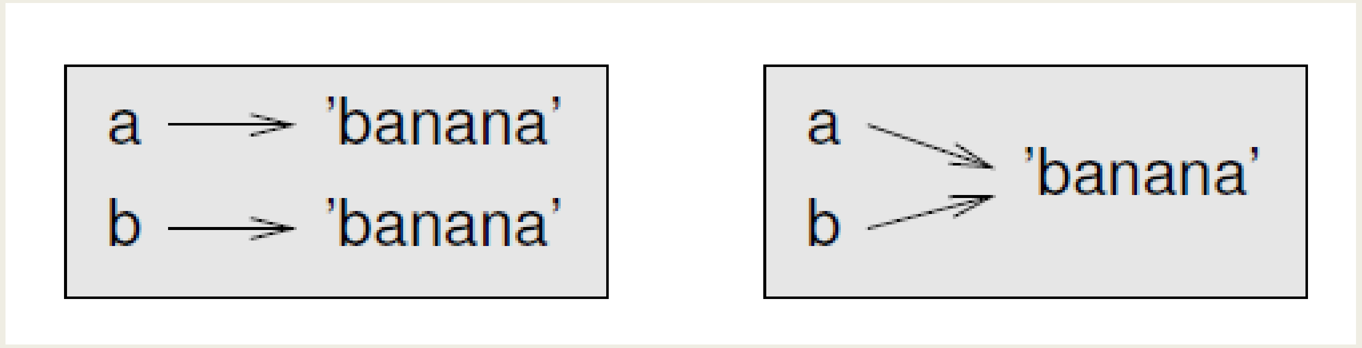
- *is* operator



```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
True
```

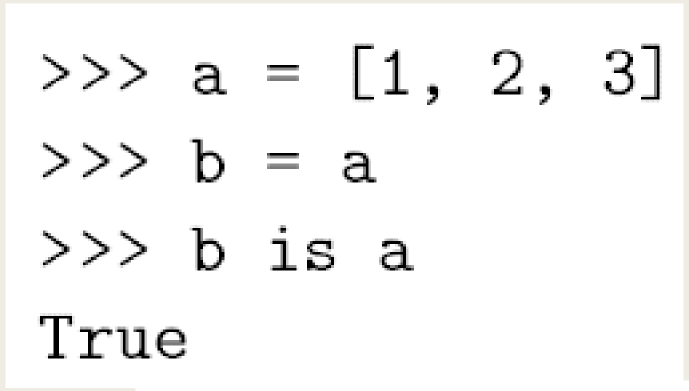


```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```

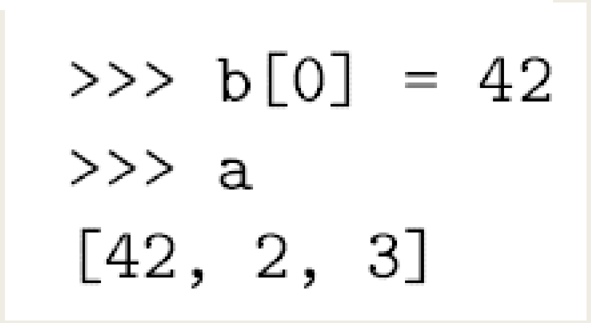


Aliasing

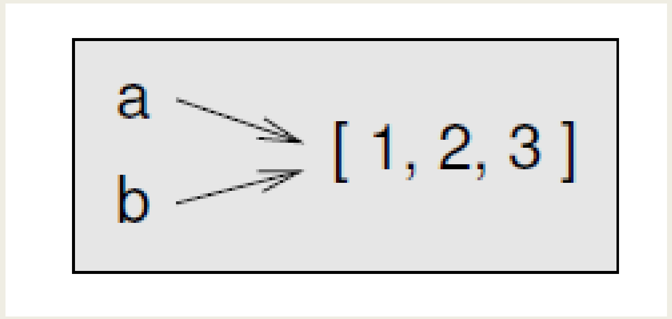
- Two *references* to the same object



```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```



```
>>> b[0] = 42
>>> a
[42, 2, 3]
```


$$\begin{tikzcd}
a \arrow[dr] & \\
 & {[\,1, 2, 3\,]} \\
b \arrow[ur] &
\end{tikzcd}$$

Example: Pass-by-reference

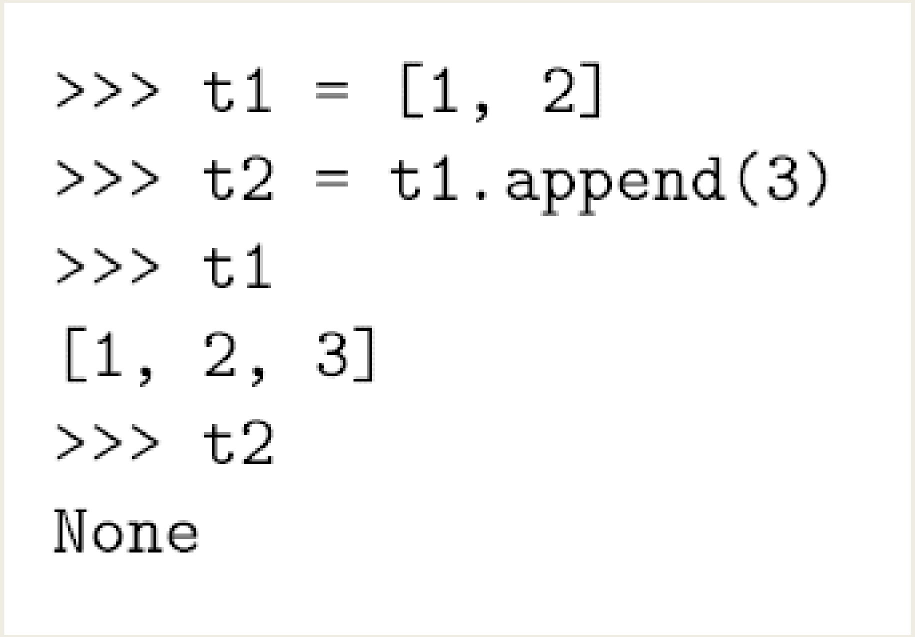
- When you *pass* a list to a function, the function gets a *reference* to the list. If the function modifies the list, the caller sees the change.

```
def delete_head(t):
    del t[0]
```

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> letters
['b', 'c']
```

Example: return None

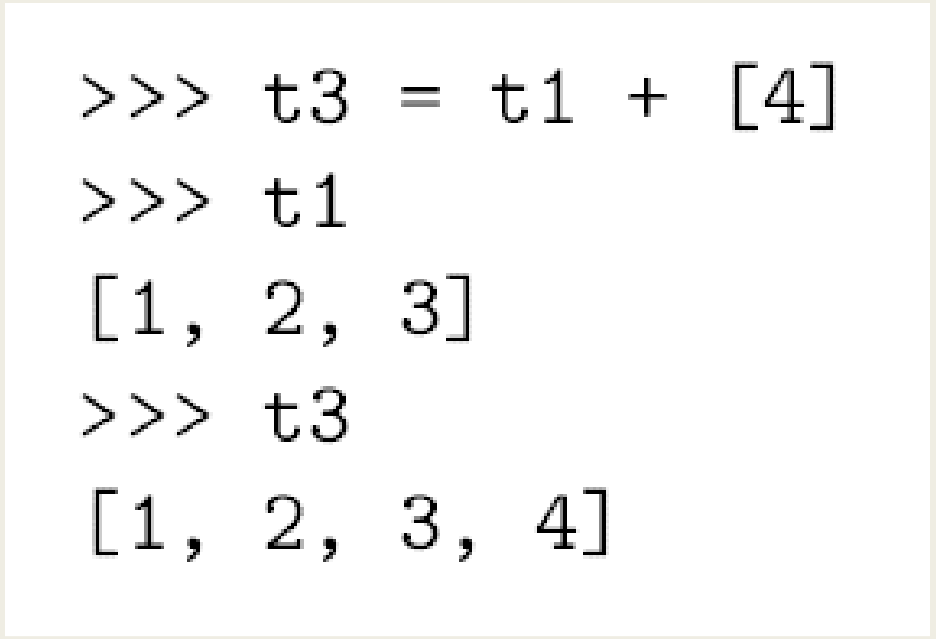
- *Append* method modifies a list
  - *Return None*



```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> t1
[1, 2, 3]
>>> t2
None
```

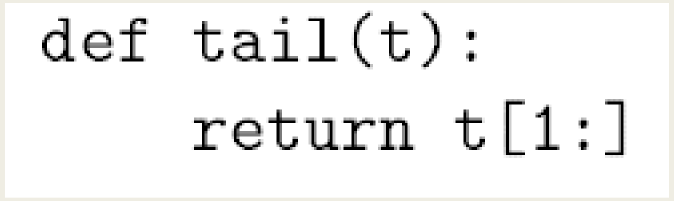
Example: +

- +
  - *creates a new list*
  - *The original list is unchanged*

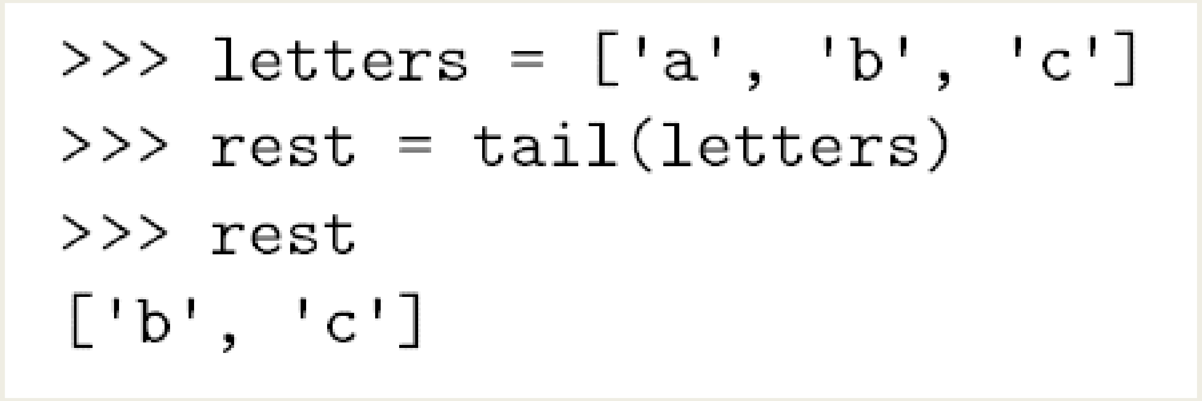


```
>>> t3 = t1 + [4]
>>> t1
[1, 2, 3]
>>> t3
[1, 2, 3, 4]
```

Example: return a new list



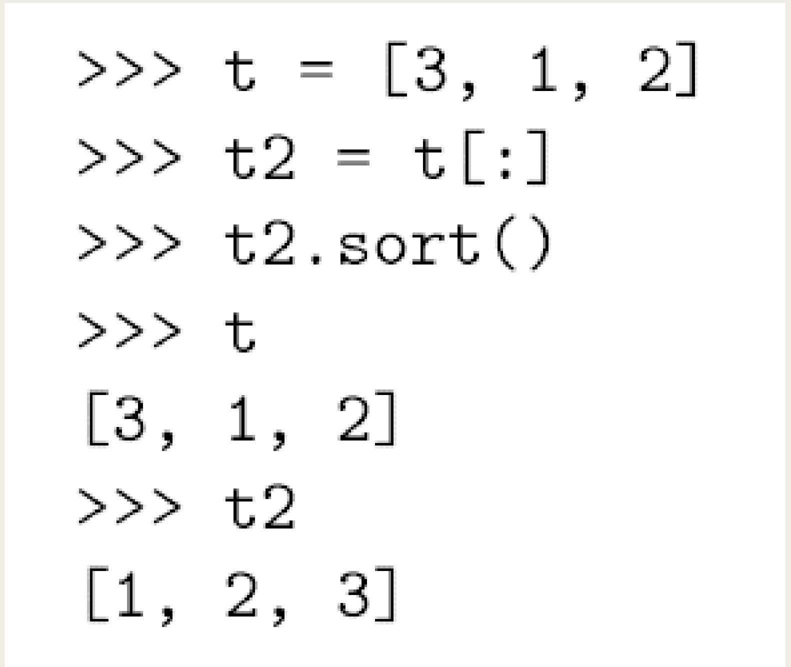
```
def tail(t):
    return t[1:]
```



```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> rest
['b', 'c']
```

Tips: debugging list operations

- Most list methods modify the argument and return *None*.
  - *String methods are different*
- Pick an idiom and stick with it.
  - *There are many ways to do it*
  - *Make sure you use it correctly (try it carefully)*
- Make copies to avoid aliasing.



```
>>> t = [3, 1, 2]
>>> t2 = t[:]
>>> t2.sort()
>>> t
[3, 1, 2]
>>> t2
[1, 2, 3]
```

Reading

- Chapter 10 in textbook “Think Python”