Basic data types

Amit Kotlovski
Open-Source Consultancy
http://amitkot.com

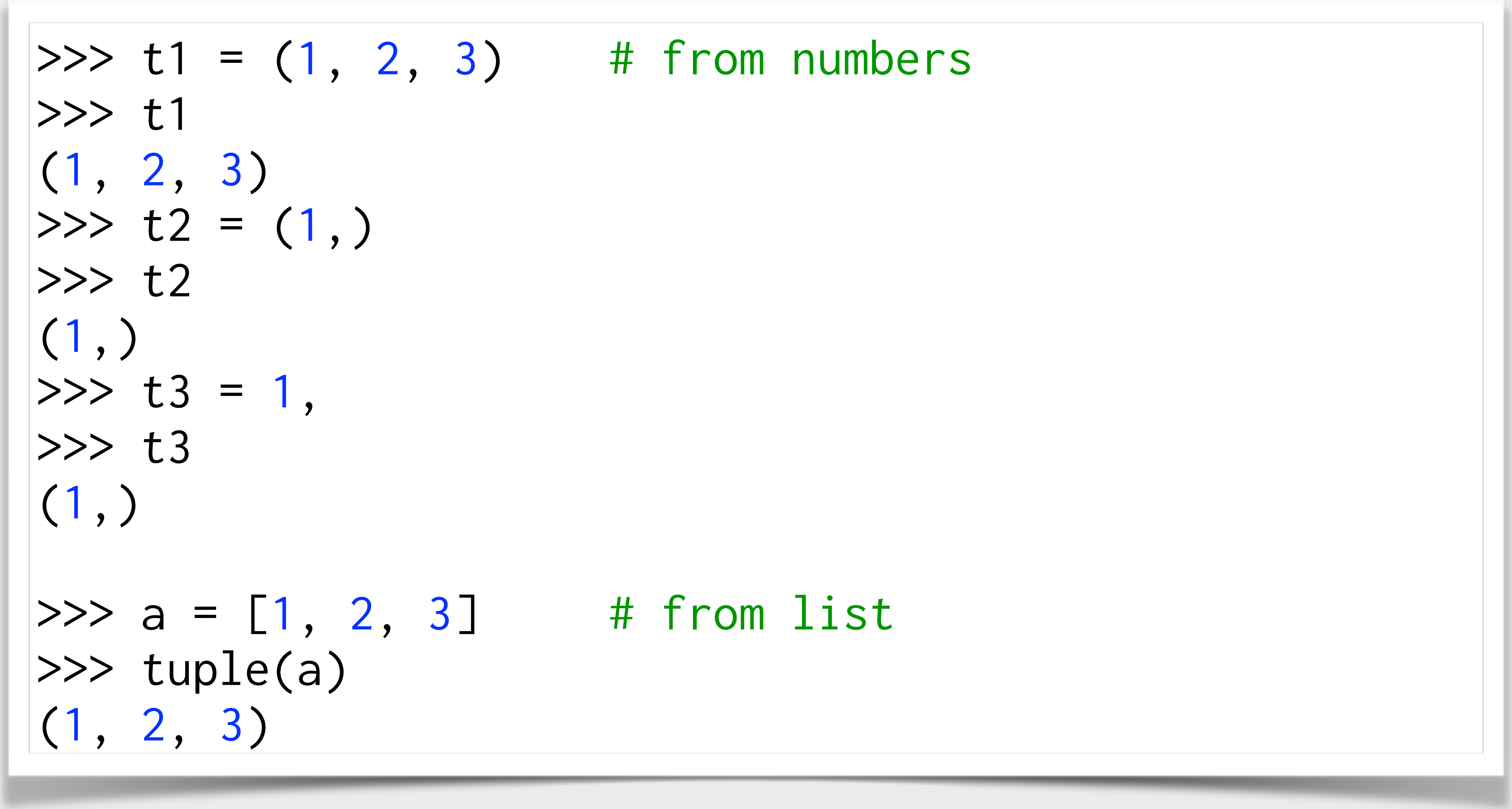
Agenda

- tuple
- list
- set
- dict
- string

tuple

- Holds a sequence of items
- Similar to list but immutable

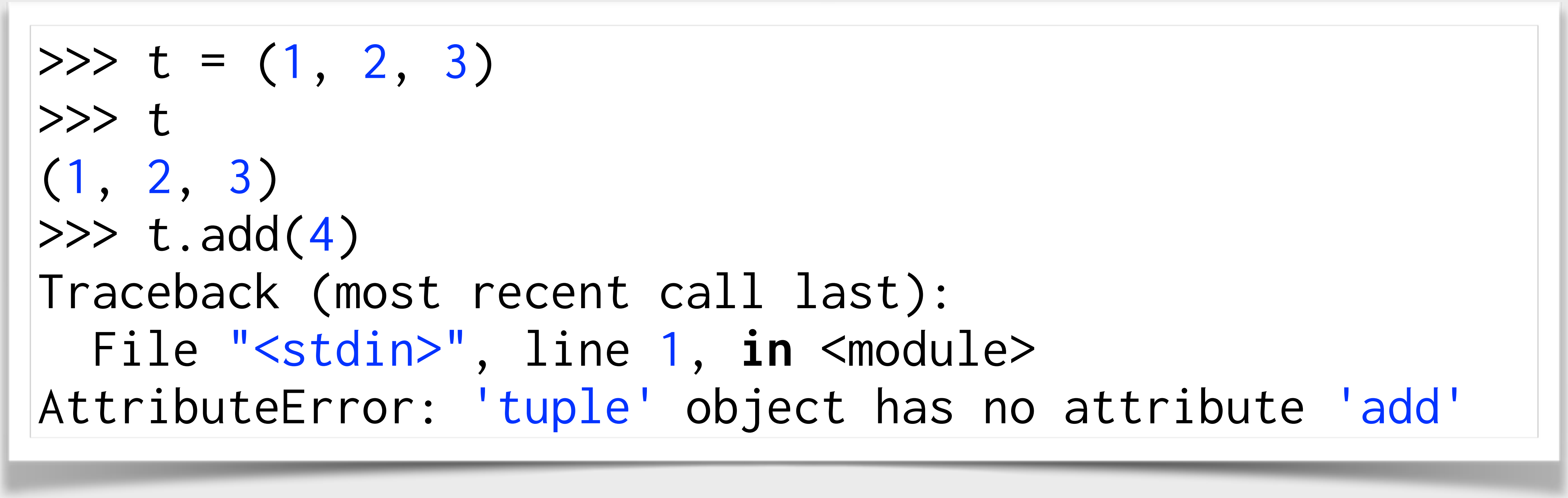
Creating a tuple



```
>>> t1 = (1, 2, 3)    # from numbers
>>> t1
(1, 2, 3)
>>> t2 = (1,)
>>> t2
(1,)
>>> t3 = 1,
>>> t3
(1,)

>>> a = [1, 2, 3]     # from list
>>> tuple(a)
(1, 2, 3)
```

Immutable container

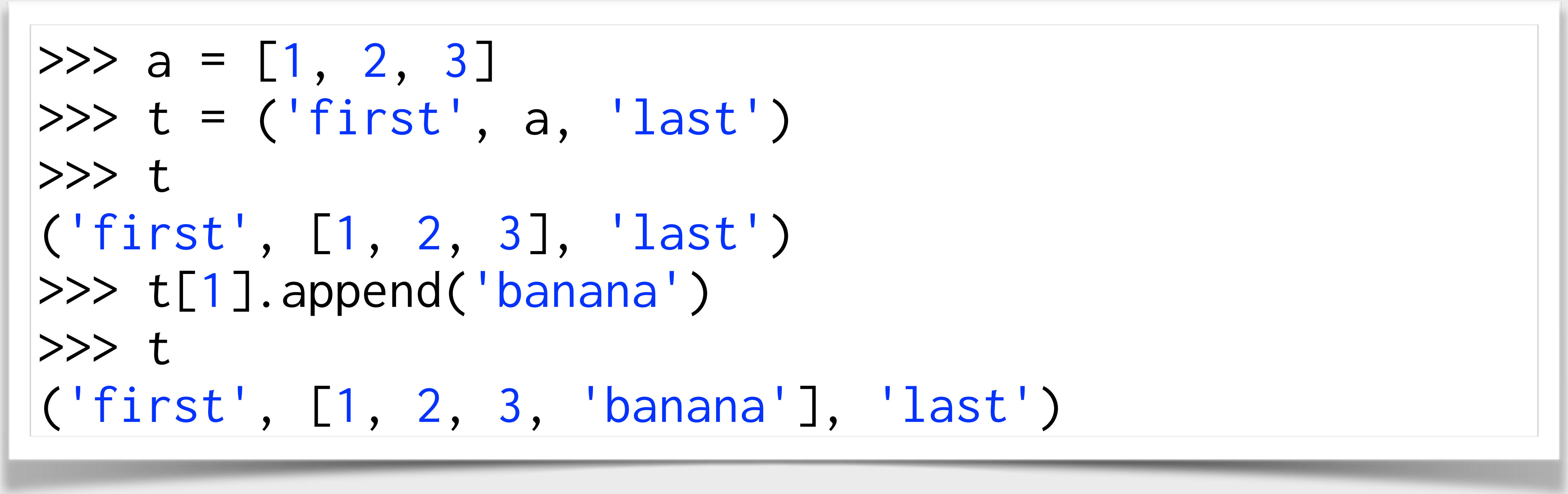
Once created, a tuple does not change



```
>>> t = (1, 2, 3)
>>> t
(1, 2, 3)
>>> t.add(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'add'
```

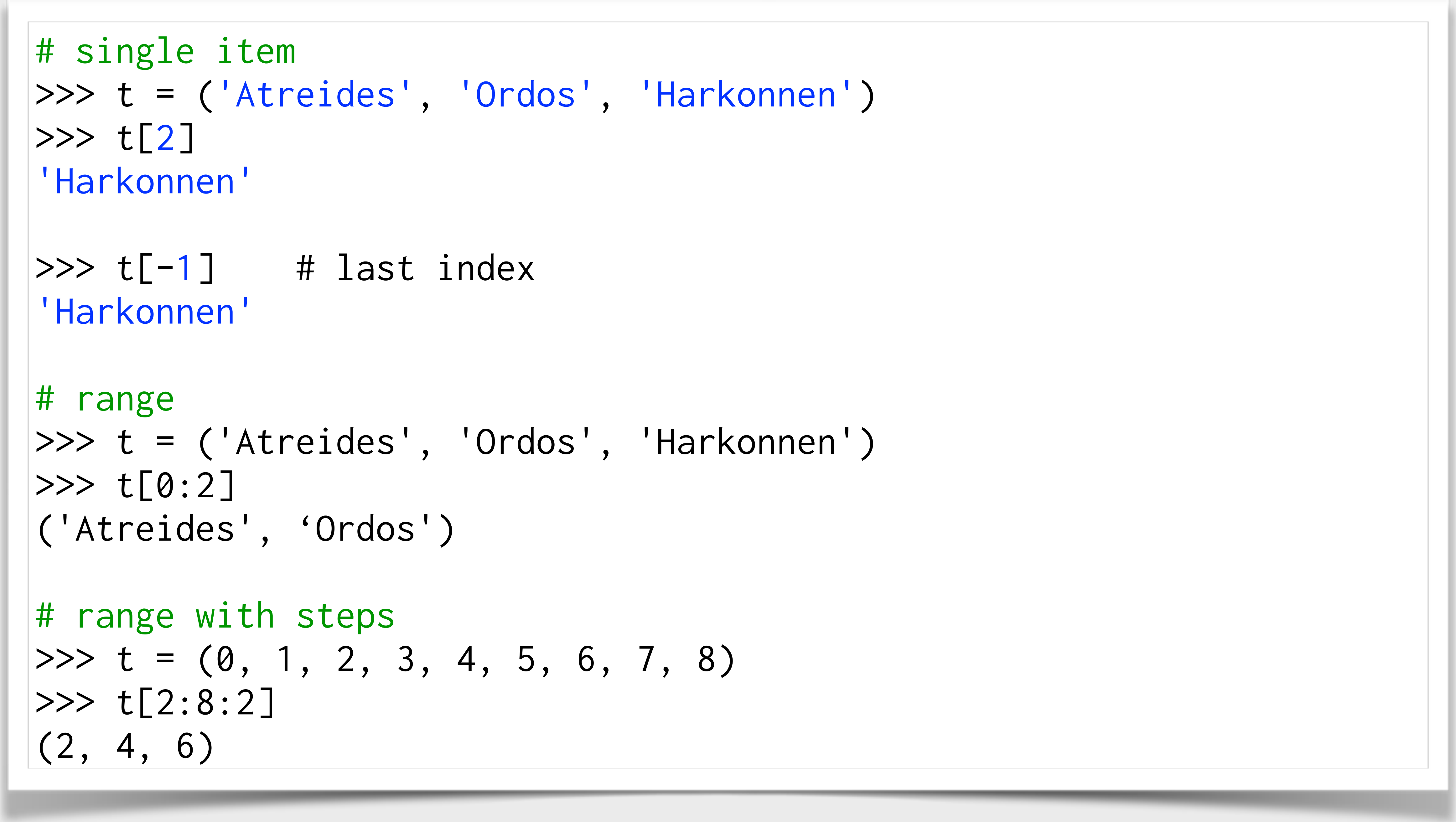
“Cheating” immutability

Keeping a mutable object inside a tuple allows mutations of that object



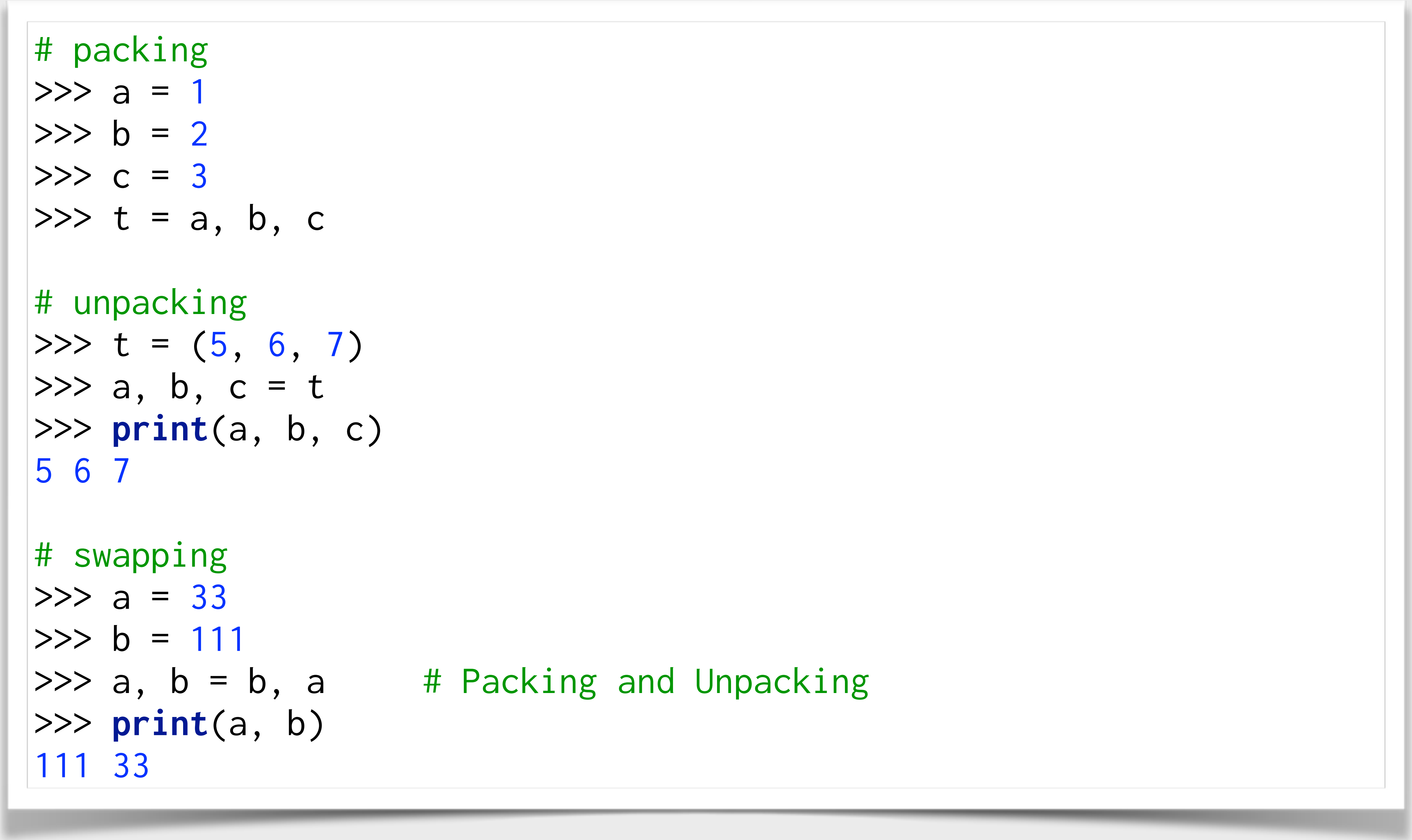
```
>>> a = [1, 2, 3]
>>> t = ('first', a, 'last')
>>> t
('first', [1, 2, 3], 'last')
>>> t[1].append('banana')
>>> t
('first', [1, 2, 3, 'banana'], 'last')
```

Slicing



```
# single item
>>> t = ('Atreides', 'Ordos', 'Harkonnen')
>>> t[2]
'Harkonnen'

>>> t[-1]    # last index
'Harkonnen'

# range
>>> t = ('Atreides', 'Ordos', 'Harkonnen')
>>> t[0:2]
('Atreides', ‘Ordos')

# range with steps
>>> t = (0, 1, 2, 3, 4, 5, 6, 7, 8)
>>> t[2:8:2]
(2, 4, 6)
```

Packing and unpacking



```
# packing
>>> a = 1
>>> b = 2
>>> c = 3
>>> t = a, b, c

# unpacking
>>> t = (5, 6, 7)
>>> a, b, c = t
>>> print(a, b, c)
5 6 7

# swapping
>>> a = 33
>>> b = 111
>>> a, b = b, a      # Packing and Unpacking
>>> print(a, b)
111 33
```

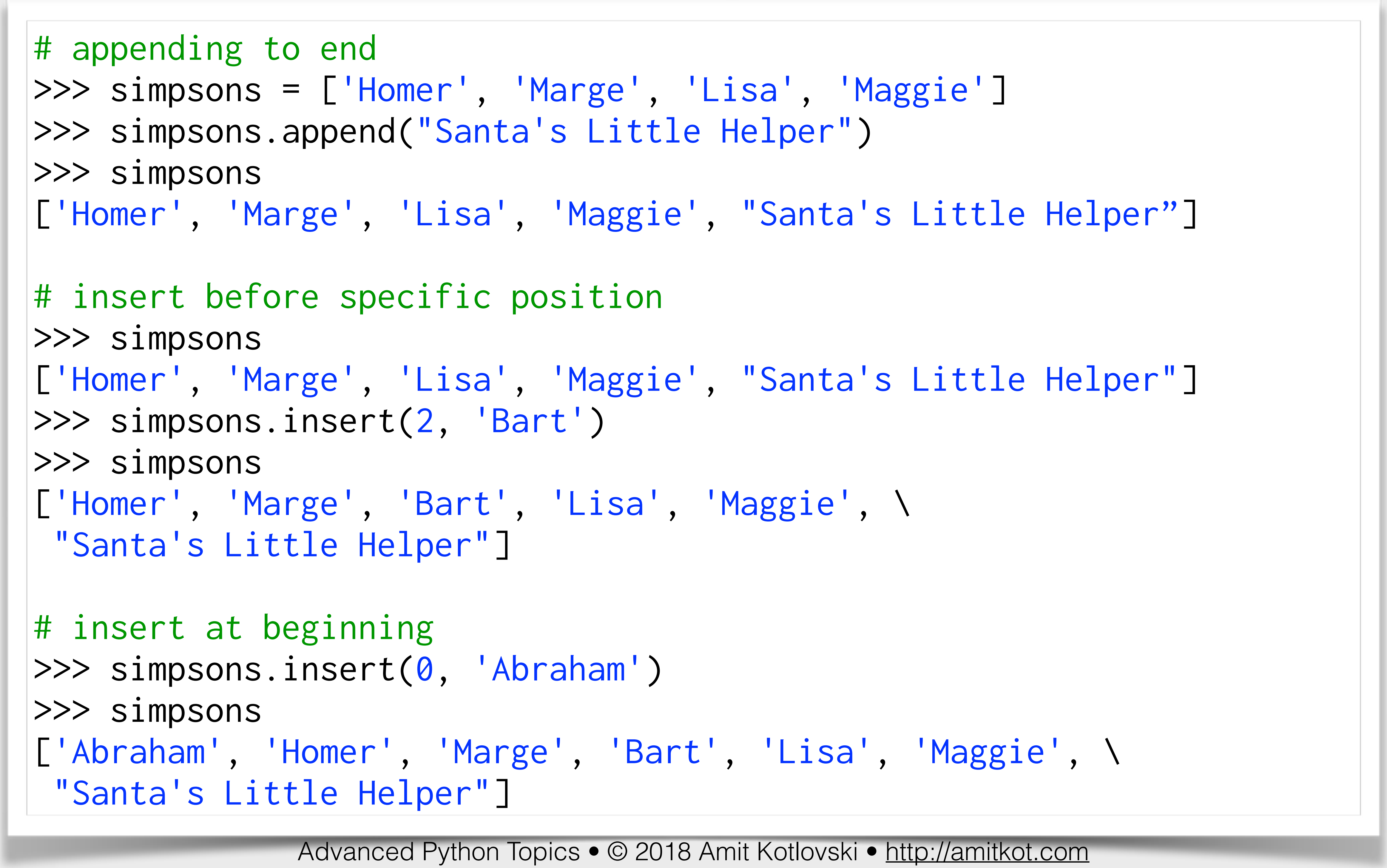
Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

list

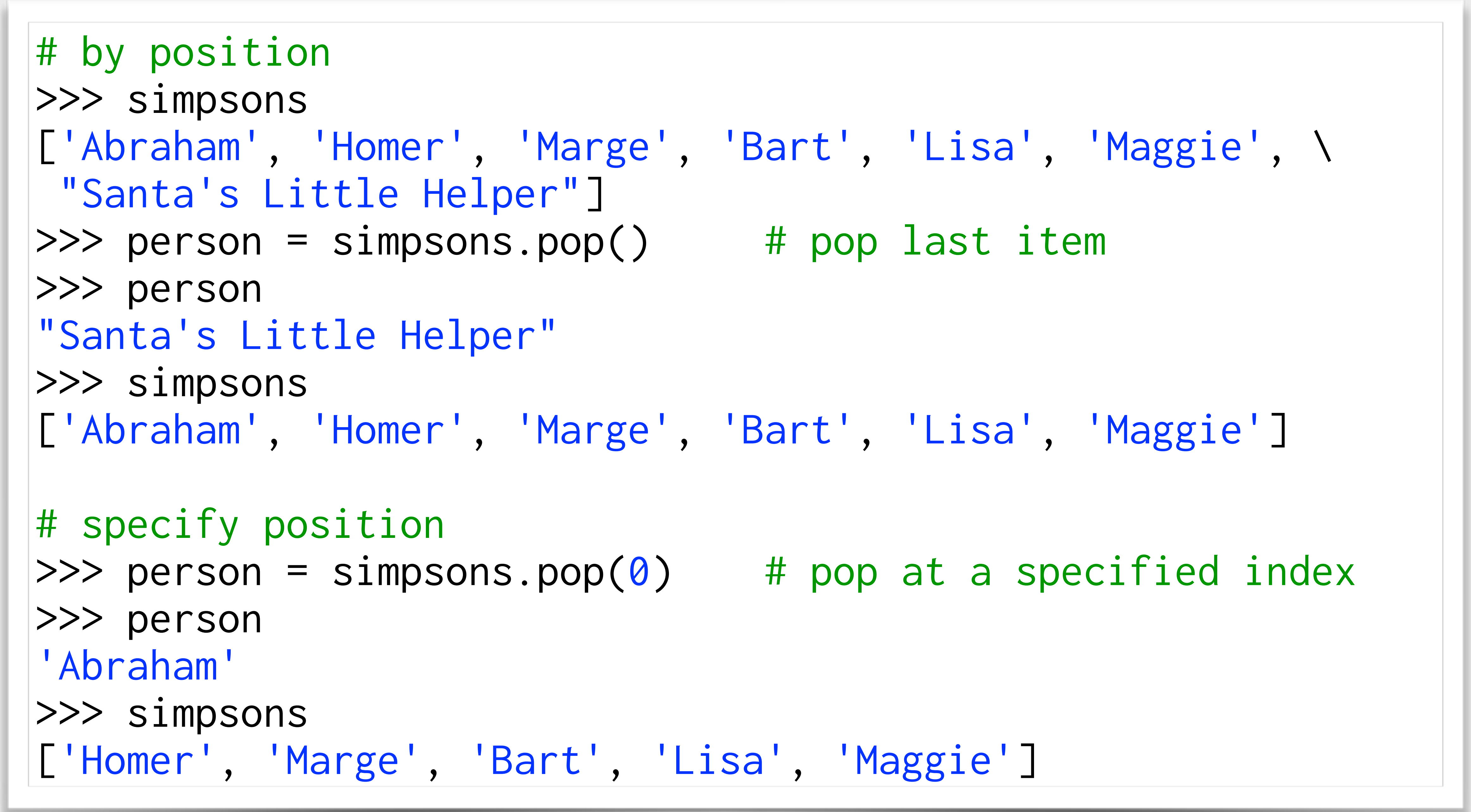
- A mutable sequence of items
- Items can be of different types

Adding items



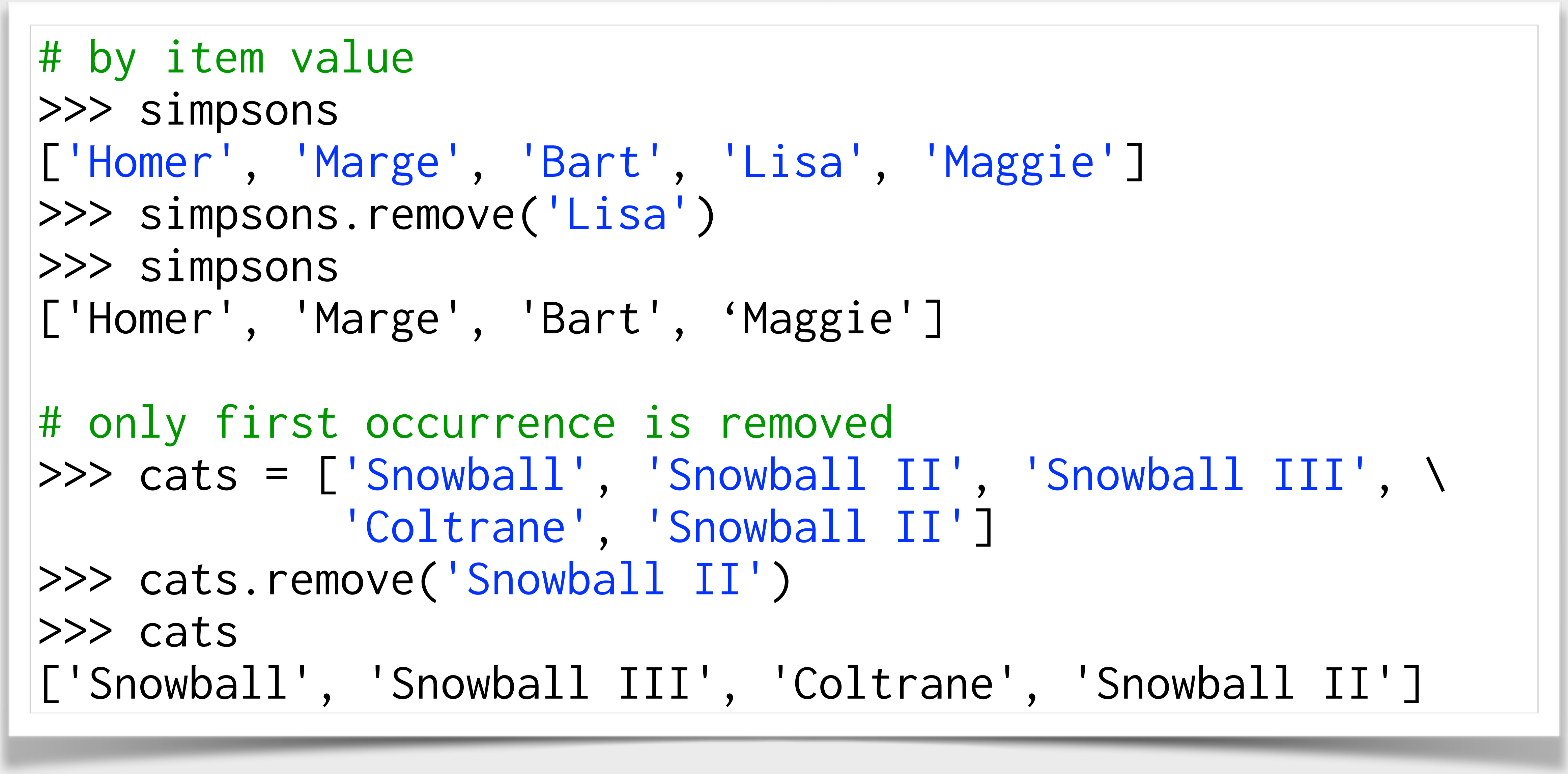
```
# appending to end
>>> simpsons = ['Homer', 'Marge', 'Lisa', 'Maggie']
>>> simpsons.append("Santa's Little Helper")
>>> simpsons
['Homer', 'Marge', 'Lisa', 'Maggie', "Santa's Little Helper”]

# insert before specific position
>>> simpsons
['Homer', 'Marge', 'Lisa', 'Maggie', "Santa's Little Helper"]
>>> simpsons.insert(2, 'Bart')
>>> simpsons
['Homer', 'Marge', 'Bart', 'Lisa', 'Maggie', \
 "Santa's Little Helper"]

# insert at beginning
>>> simpsons.insert(0, 'Abraham')
>>> simpsons
['Abraham', 'Homer', 'Marge', 'Bart', 'Lisa', 'Maggie', \
 "Santa's Little Helper"]
```

Removing items



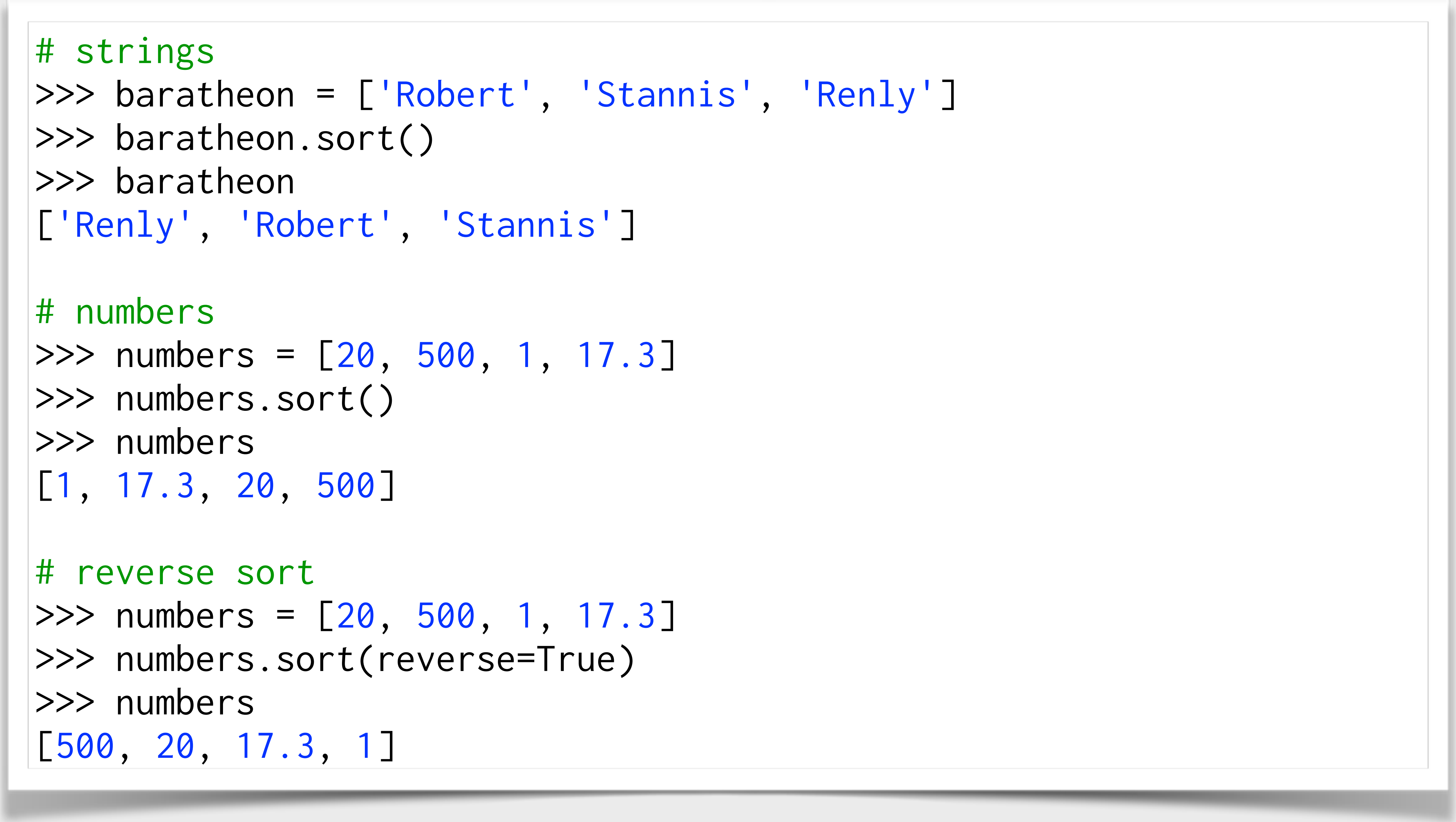
```
# by position
>>> simpsons
['Abraham', 'Homer', 'Marge', 'Bart', 'Lisa', 'Maggie', \
 "Santa's Little Helper"]
>>> person = simpsons.pop()     # pop last item
>>> person
"Santa's Little Helper"
>>> simpsons
['Abraham', 'Homer', 'Marge', 'Bart', 'Lisa', 'Maggie']

# specify position
>>> person = simpsons.pop(0)    # pop at a specified index
>>> person
'Abraham'
>>> simpsons
['Homer', 'Marge', 'Bart', 'Lisa', 'Maggie']
```

Removing items



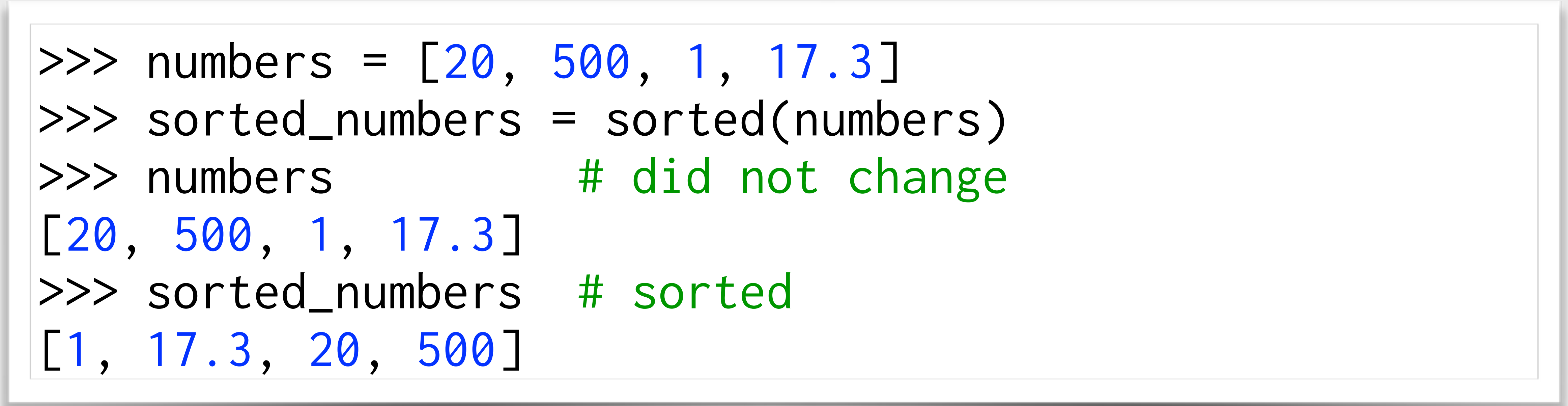
```
# by item value
>>> simpsons
['Homer', 'Marge', 'Bart', 'Lisa', 'Maggie']
>>> simpsons.remove('Lisa')
>>> simpsons
['Homer', 'Marge', 'Bart', ‘Maggie']

# only first occurrence is removed
>>> cats = ['Snowball', 'Snowball II', 'Snowball III', \
            'Coltrane', 'Snowball II']
>>> cats.remove('Snowball II')
>>> cats
['Snowball', 'Snowball III', 'Coltrane', 'Snowball II']
```

Sort - in place



```
# strings
>>> baratheon = ['Robert', 'Stannis', 'Renly']
>>> baratheon.sort()
>>> baratheon
['Renly', 'Robert', 'Stannis']

# numbers
>>> numbers = [20, 500, 1, 17.3]
>>> numbers.sort()
>>> numbers
[1, 17.3, 20, 500]

# reverse sort
>>> numbers = [20, 500, 1, 17.3]
>>> numbers.sort(reverse=True)
>>> numbers
[500, 20, 17.3, 1]
```

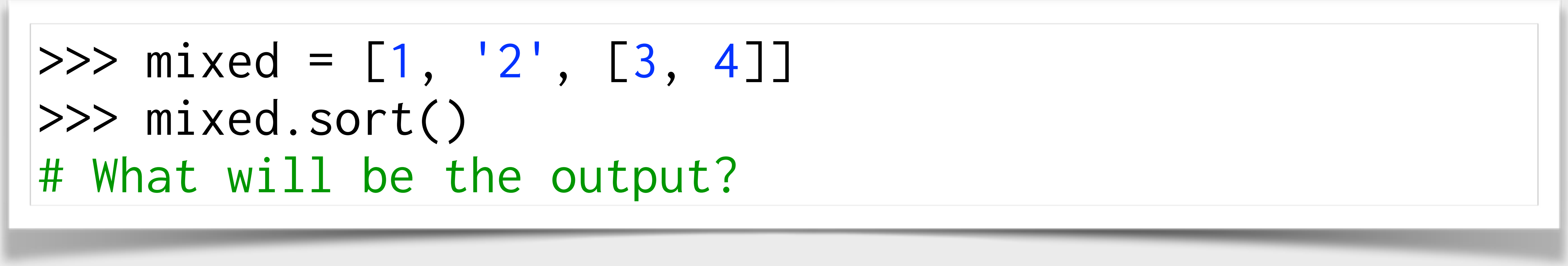
Sort - not in place



```
>>> numbers = [20, 500, 1, 17.3]
>>> sorted_numbers = sorted(numbers)
>>> numbers         # did not change
[20, 500, 1, 17.3]
>>> sorted_numbers  # sorted
[1, 17.3, 20, 500]
```

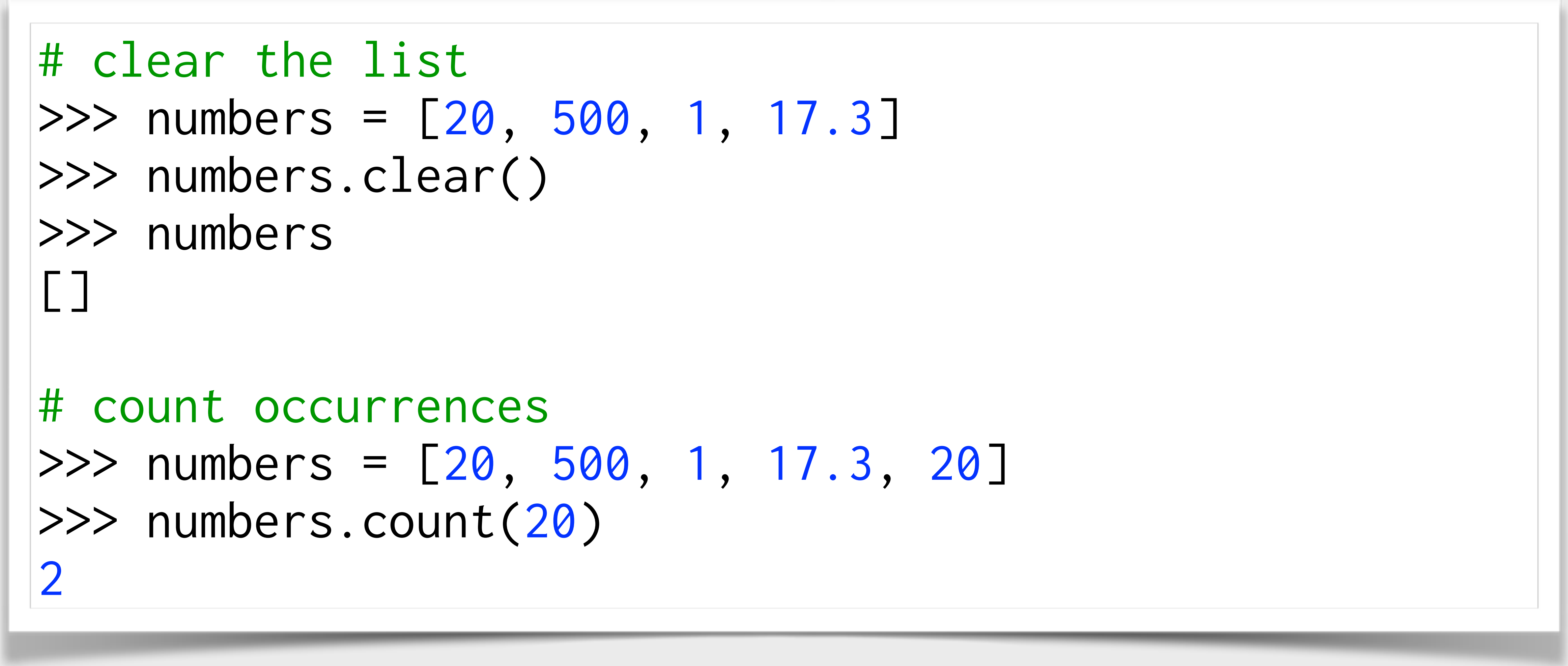
Sort - mixing types

- Python 2 - possible (horrible!)
- Python 3 - `TypeError`



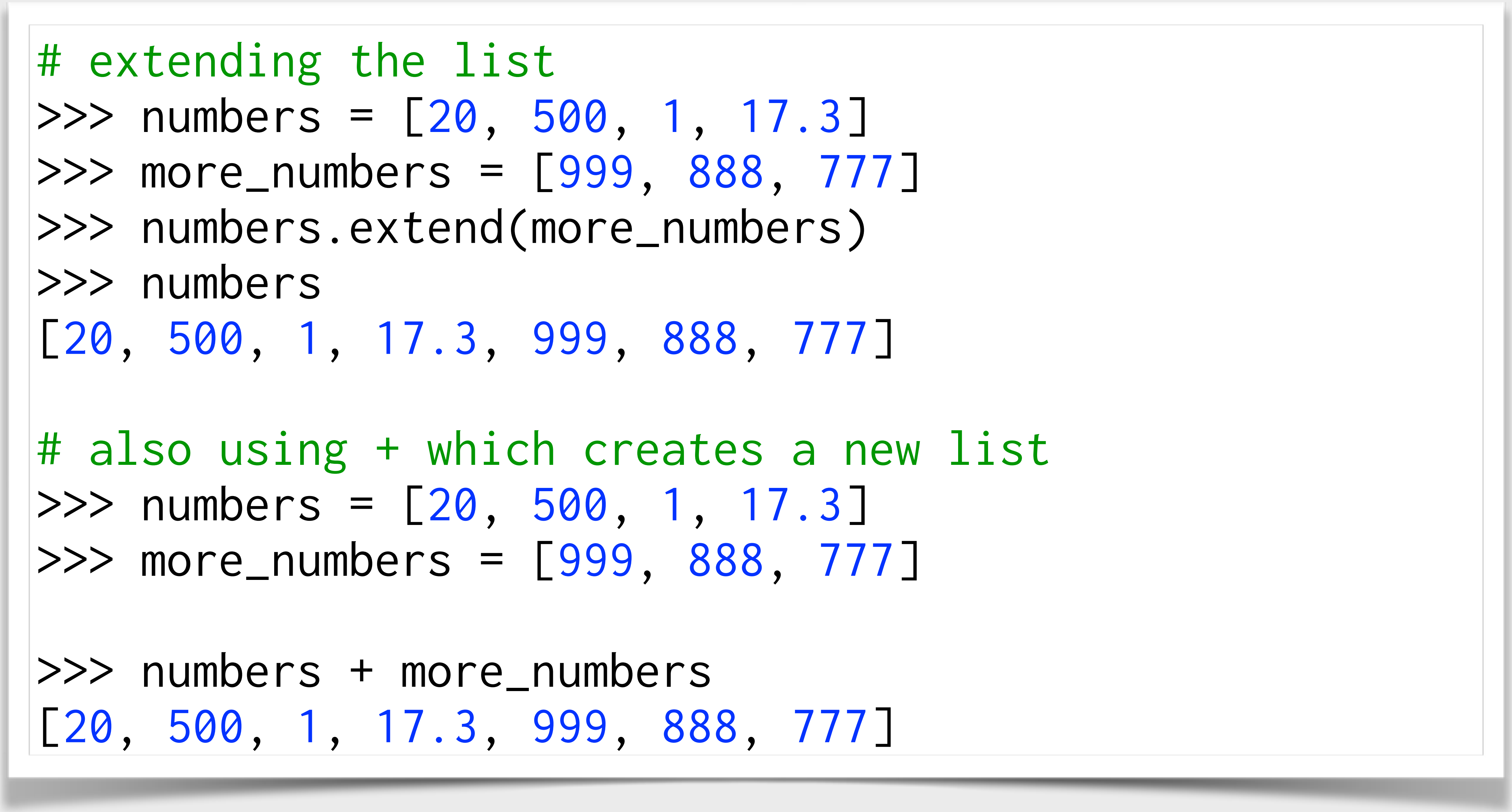
```
>>> mixed = [1, '2', [3, 4]]
>>> mixed.sort()
# What will be the output?
```

List methods



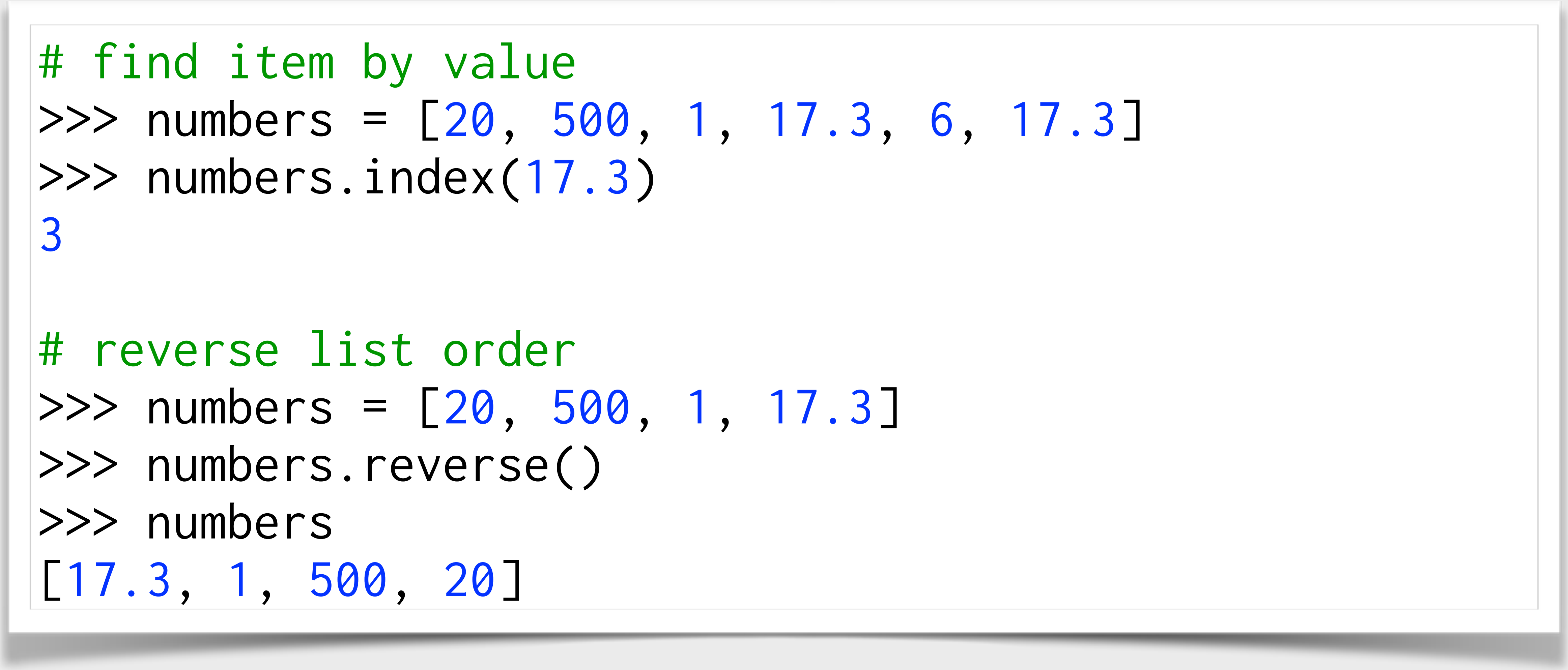
```
# clear the list
>>> numbers = [20, 500, 1, 17.3]
>>> numbers.clear()
>>> numbers
[]

# count occurrences
>>> numbers = [20, 500, 1, 17.3, 20]
>>> numbers.count(20)
2
```

List methods



```
# extending the list
>>> numbers = [20, 500, 1, 17.3]
>>> more_numbers = [999, 888, 777]
>>> numbers.extend(more_numbers)
>>> numbers
[20, 500, 1, 17.3, 999, 888, 777]

# also using + which creates a new list
>>> numbers = [20, 500, 1, 17.3]
>>> more_numbers = [999, 888, 777]

>>> numbers + more_numbers
[20, 500, 1, 17.3, 999, 888, 777]
```

List methods



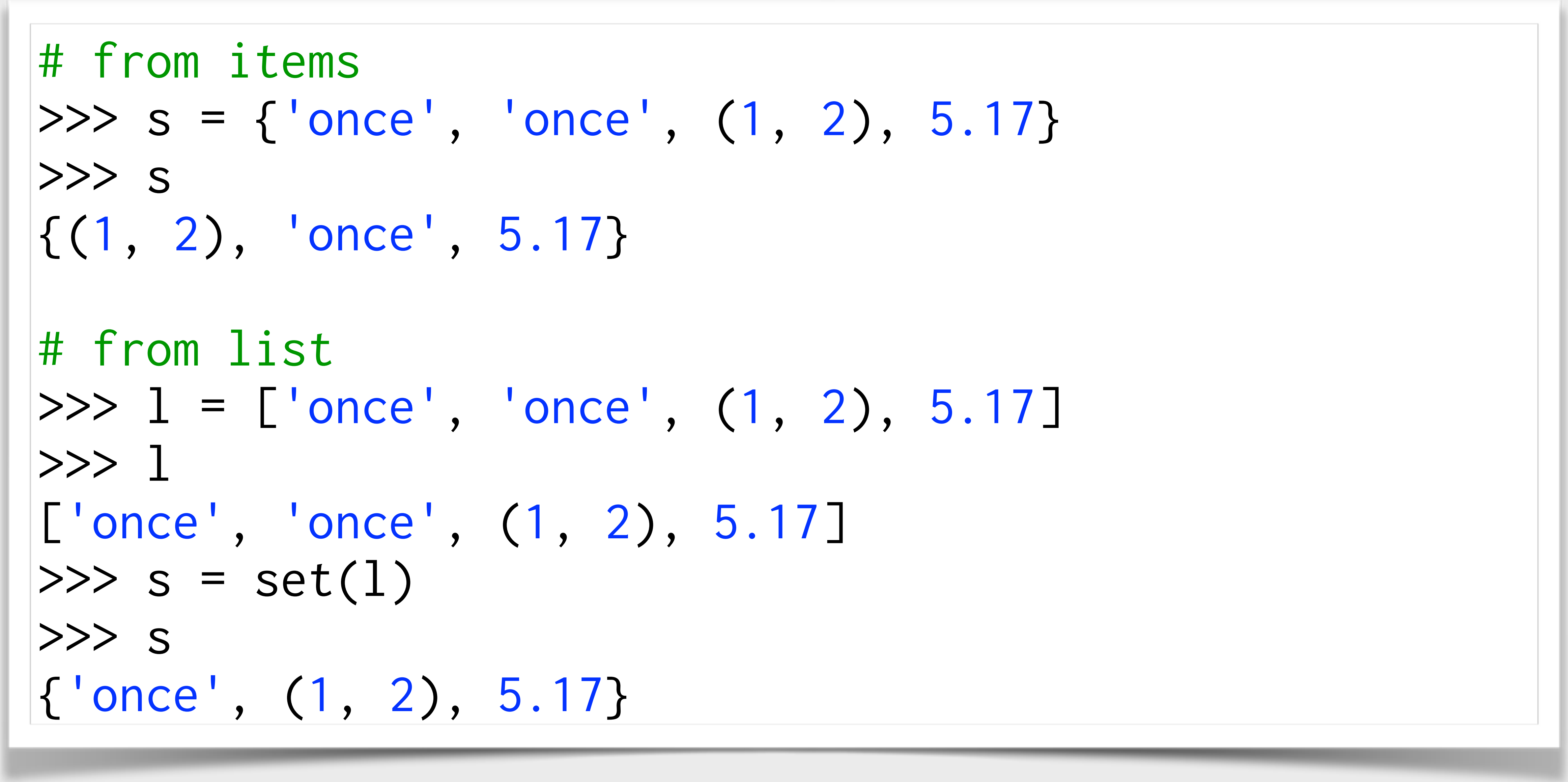
```
# find item by value
>>> numbers = [20, 500, 1, 17.3, 6, 17.3]
>>> numbers.index(17.3)
3

# reverse list order
>>> numbers = [20, 500, 1, 17.3]
>>> numbers.reverse()
>>> numbers
[17.3, 1, 500, 20]
```

Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

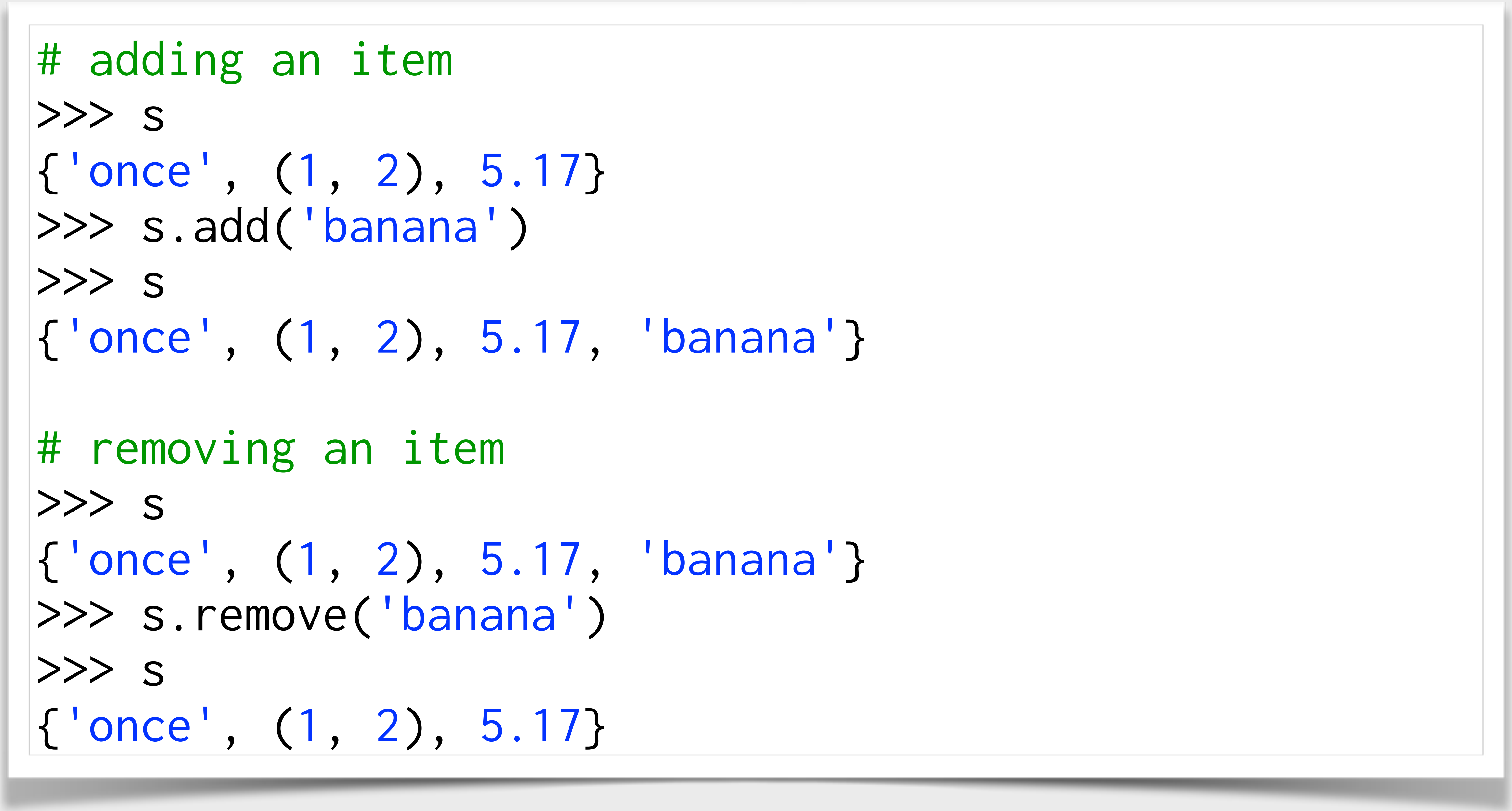
set

Create a new set



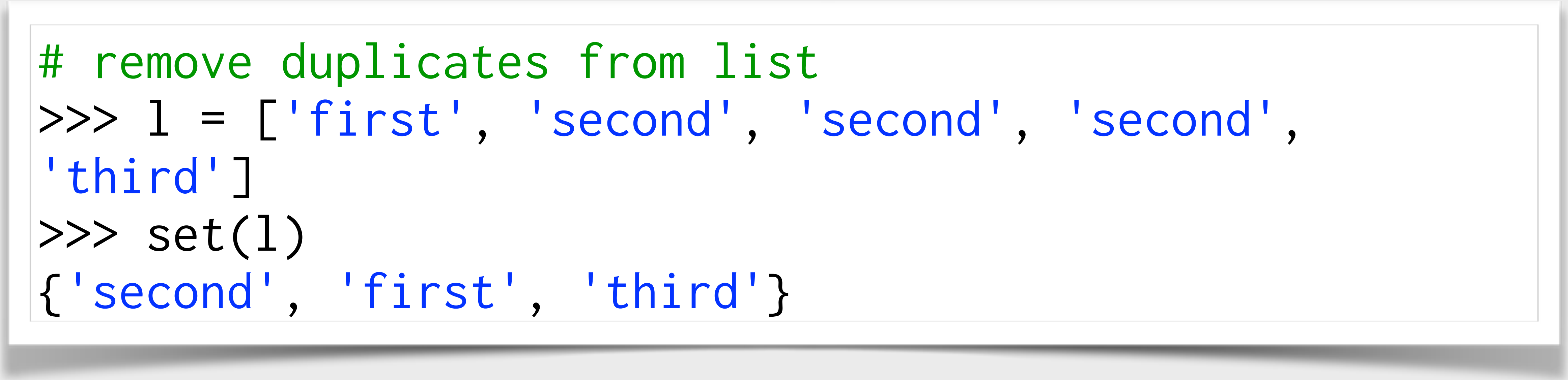
```
# from items
>>> s = {'once', 'once', (1, 2), 5.17}
>>> s
{(1, 2), 'once', 5.17}

# from list
>>> l = ['once', 'once', (1, 2), 5.17]
>>> l
['once', 'once', (1, 2), 5.17]
>>> s = set(l)
>>> s
{'once', (1, 2), 5.17}
```

Adding items



```
# adding an item
>>> s
{'once', (1, 2), 5.17}
>>> s.add('banana')
>>> s
{'once', (1, 2), 5.17, 'banana'}

# removing an item
>>> s
{'once', (1, 2), 5.17, 'banana'}
>>> s.remove('banana')
>>> s
{'once', (1, 2), 5.17}
```

Useful trick

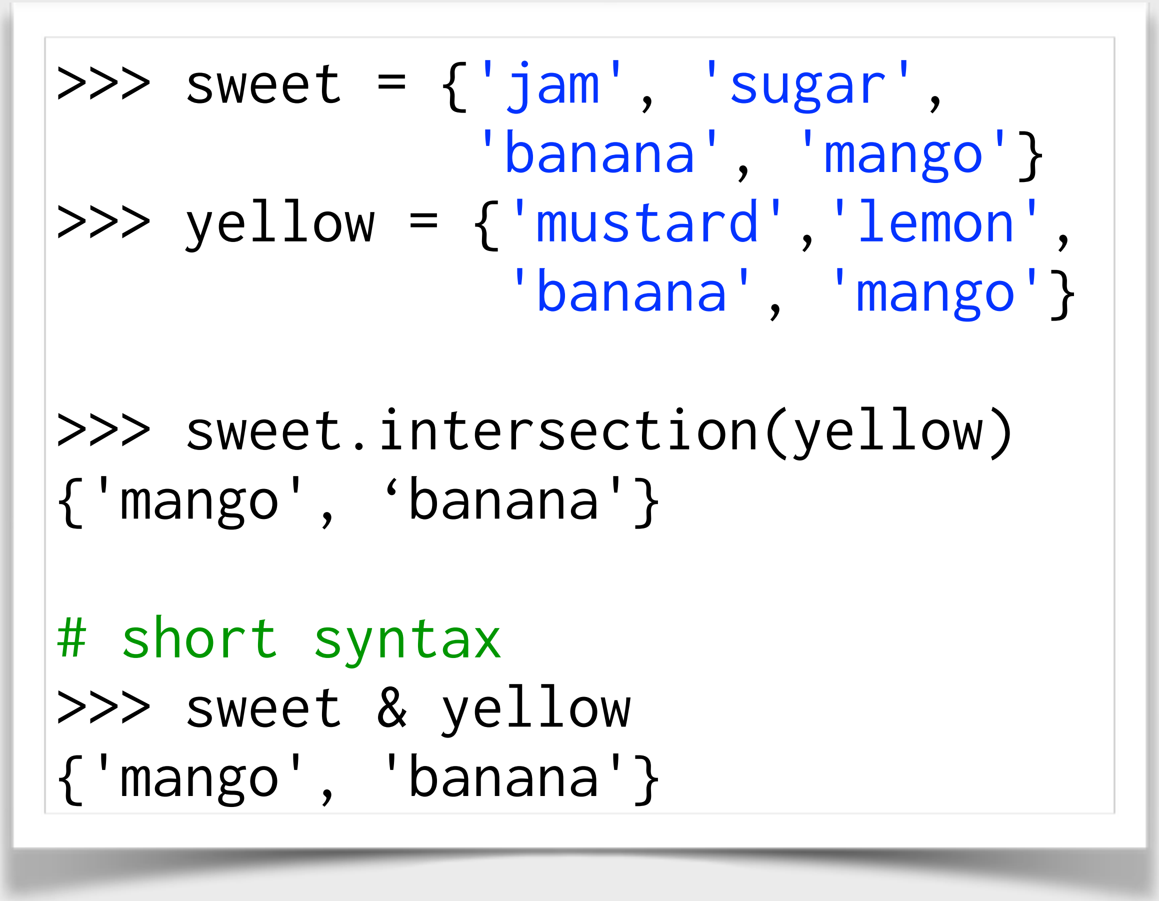


```
# remove duplicates from list
>>> l = ['first', 'second', 'second', 'second',
'third']
>>> set(l)
{'second', 'first', 'third'}
```

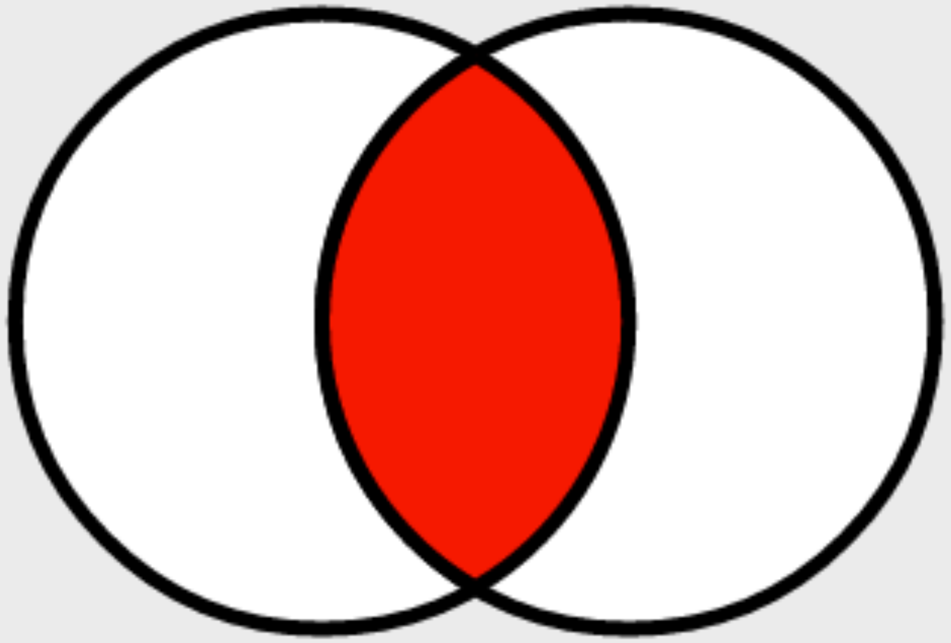
Set operations

- Sets provide a number of useful operations

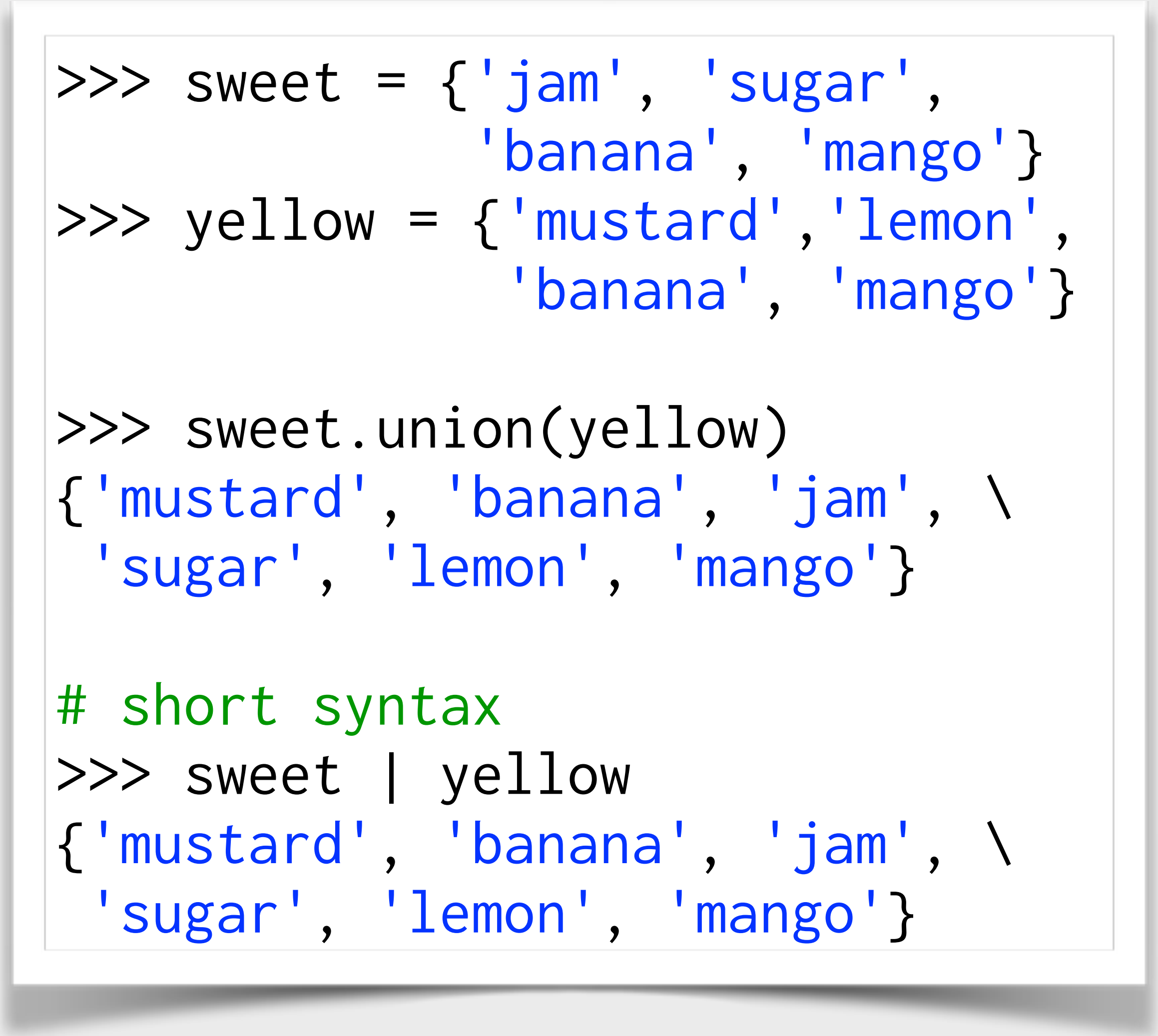
Intersection



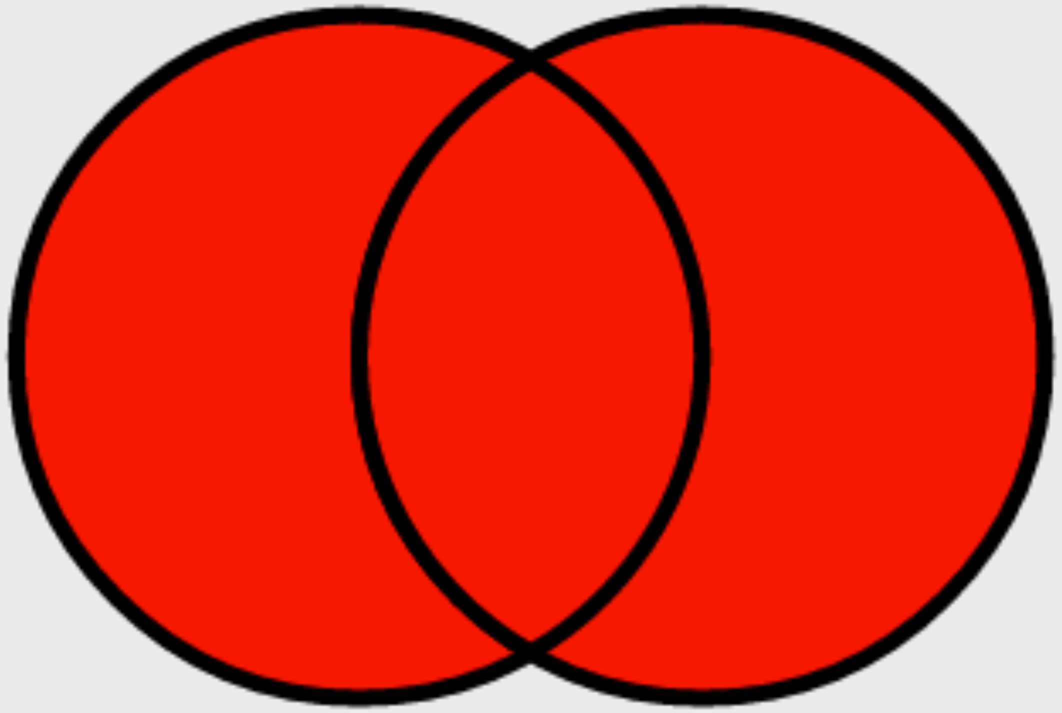
```
>>> sweet = {'jam', 'sugar',
             'banana', 'mango'}
>>> yellow = {'mustard','lemon',
              'banana', 'mango'}

>>> sweet.intersection(yellow)
{'mango', ‘banana'}

# short syntax
>>> sweet & yellow
{'mango', 'banana'}
```



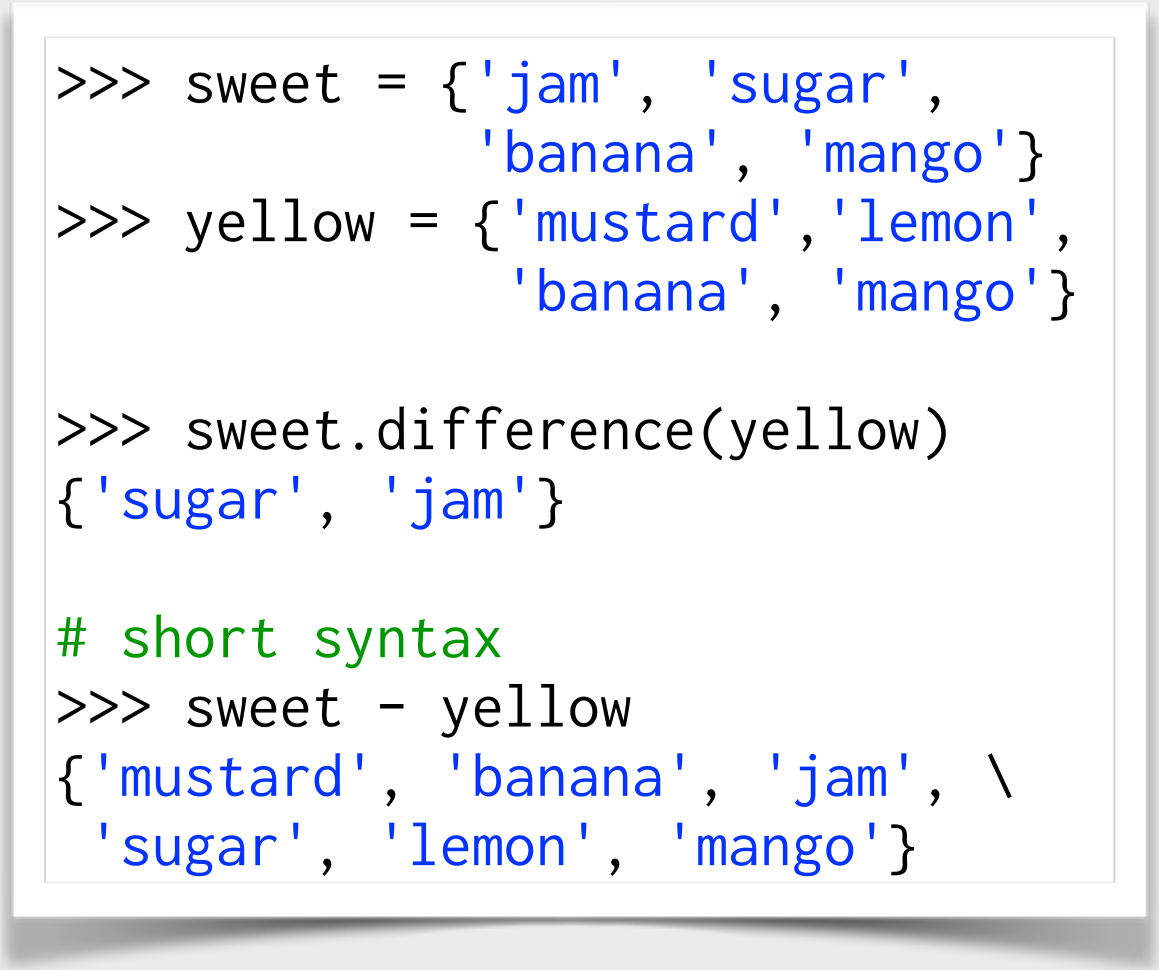
Union



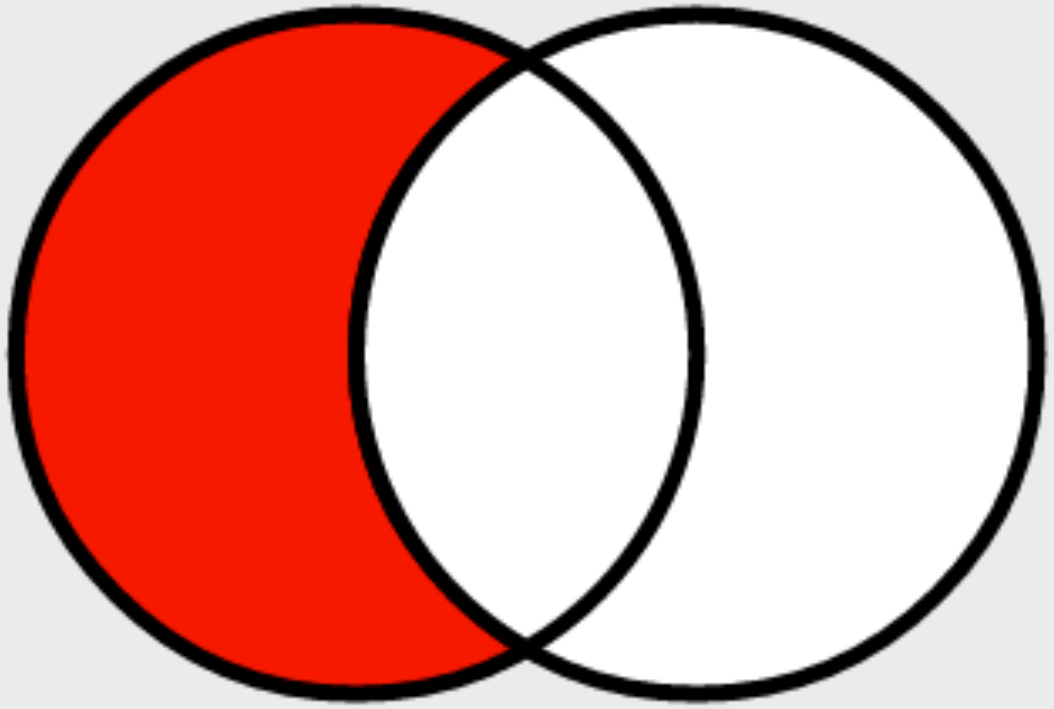
```
>>> sweet = {'jam', 'sugar',
             'banana', 'mango'}
>>> yellow = {'mustard','lemon',
              'banana', 'mango'}

>>> sweet.union(yellow)
{'mustard', 'banana', 'jam', \
 'sugar', 'lemon', 'mango'}

# short syntax
>>> sweet | yellow
{'mustard', 'banana', 'jam', \
 'sugar', 'lemon', 'mango'}
```



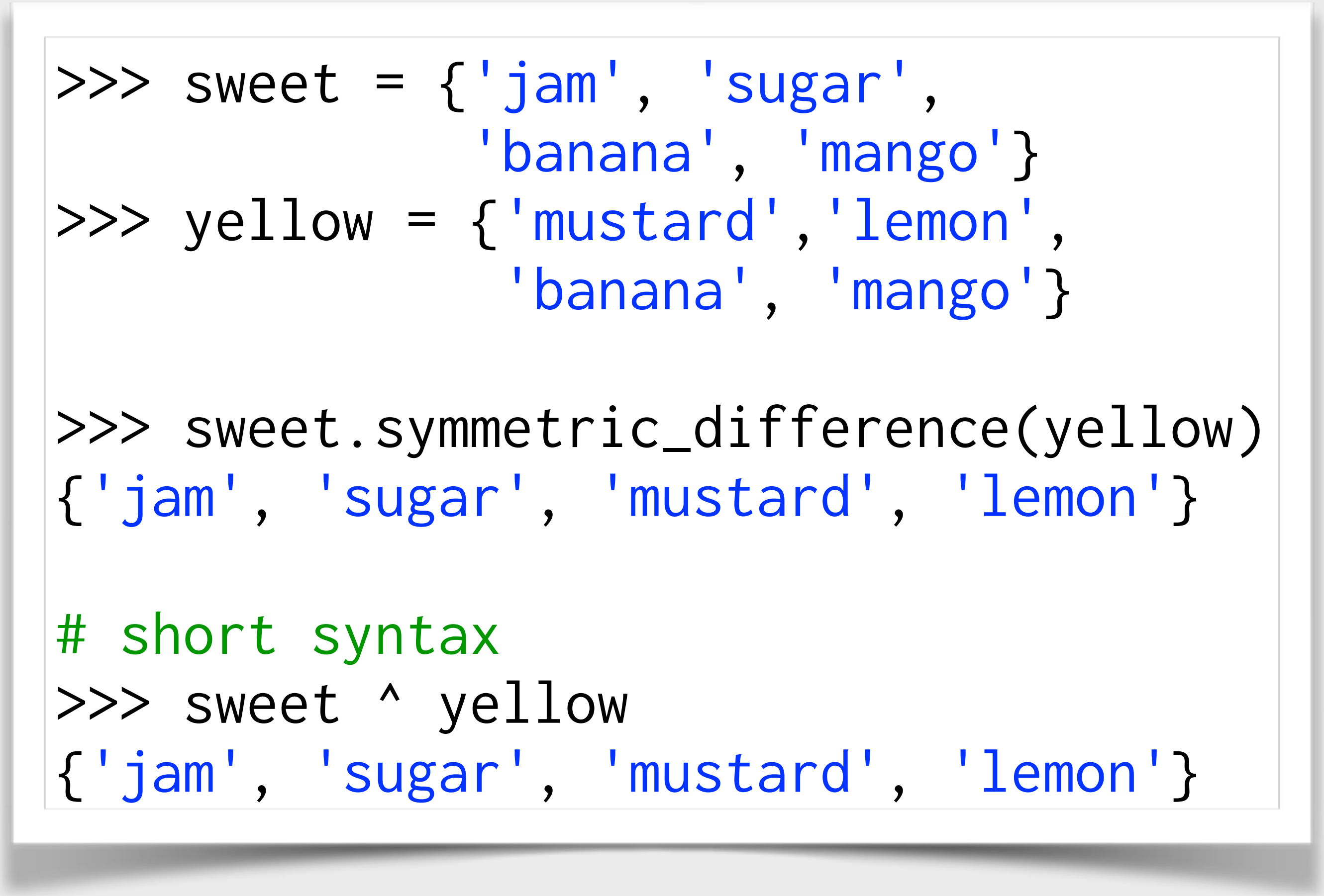
Difference



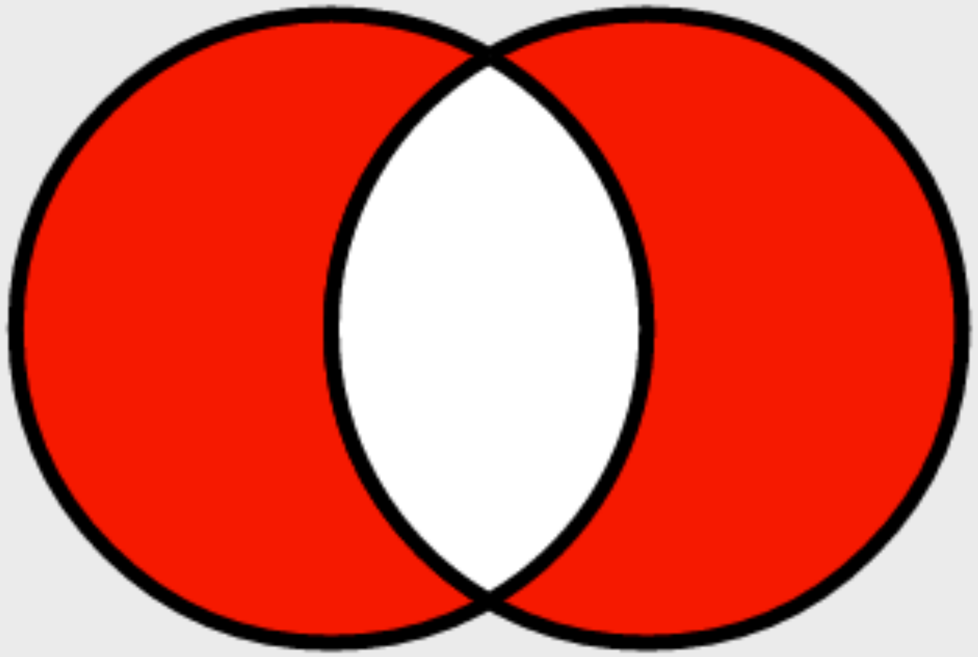
```
>>> sweet = {'jam', 'sugar',
             'banana', 'mango'}
>>> yellow = {'mustard','lemon',
              'banana', 'mango'}

>>> sweet.difference(yellow)
{'sugar', 'jam'}

# short syntax
>>> sweet - yellow
{'mustard', 'banana', 'jam', \
 'sugar', 'lemon', 'mango'}
```



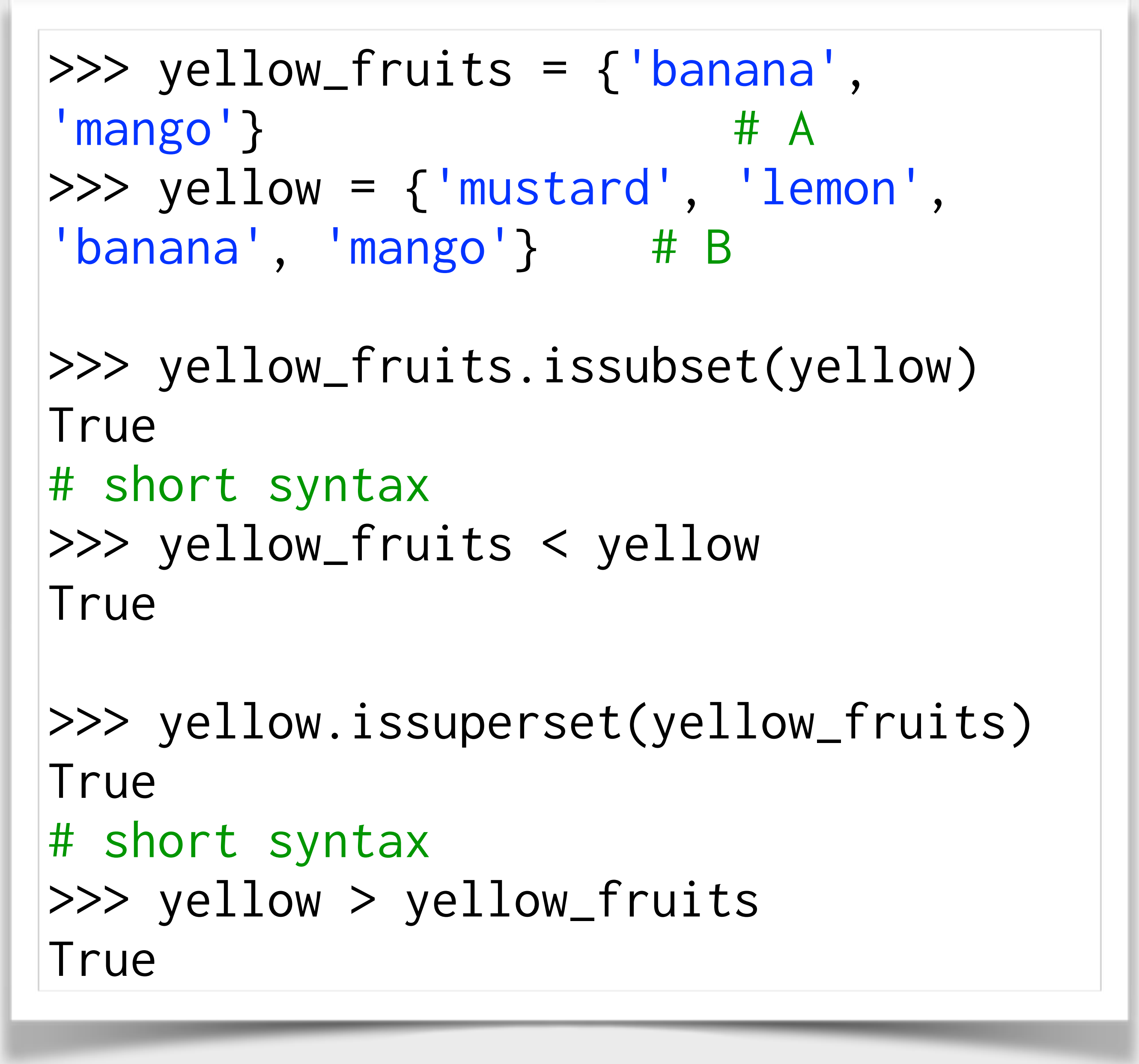
Symmetric Difference



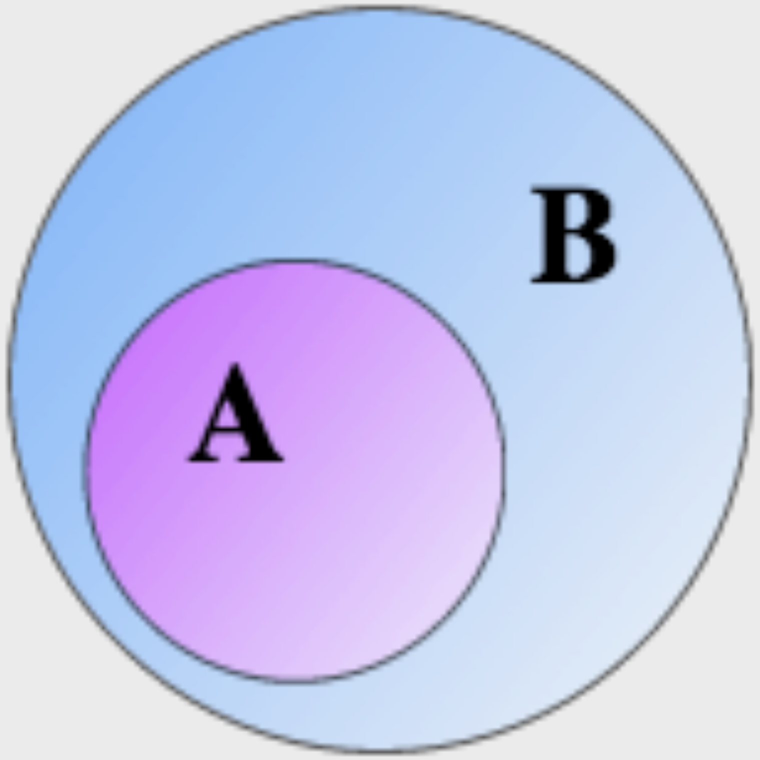
```
>>> sweet = {'jam', 'sugar',
             'banana', 'mango'}
>>> yellow = {'mustard','lemon',
              'banana', 'mango'}

>>> sweet.symmetric_difference(yellow)
{'jam', 'sugar', 'mustard', 'lemon'}

# short syntax
>>> sweet ^ yellow
{'jam', 'sugar', 'mustard', 'lemon'}
```



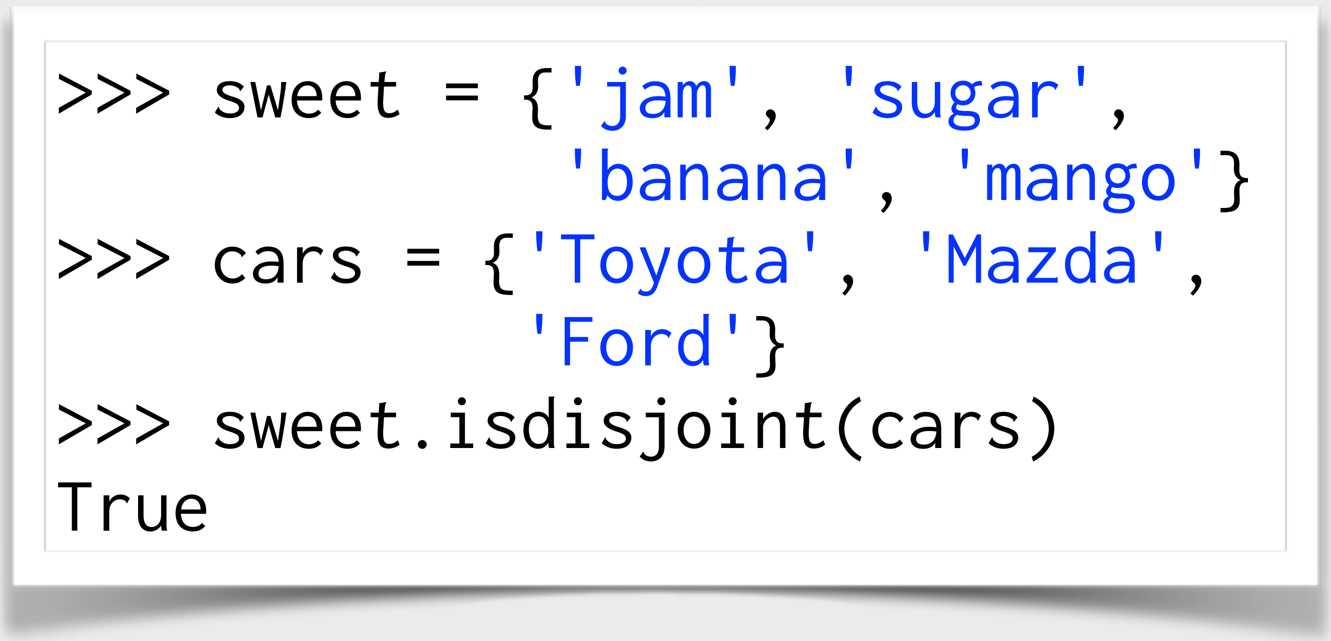
Subset / superset



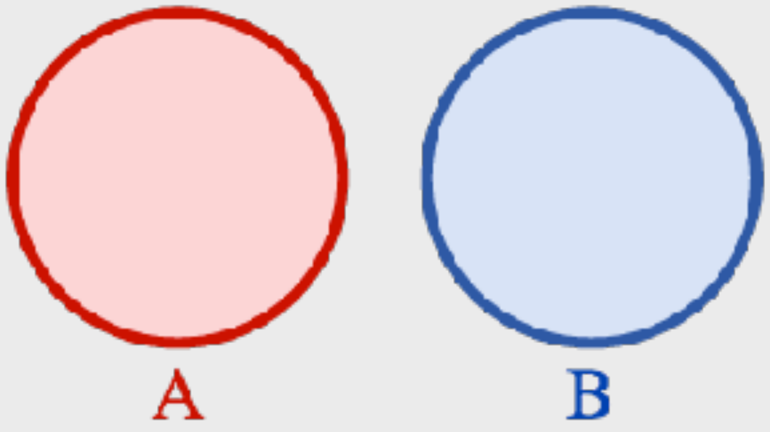
```
>>> yellow_fruits = {'banana',
'mango'}                        # A
>>> yellow = {'mustard', 'lemon',
'banana', 'mango'}     # B

>>> yellow_fruits.issubset(yellow)
True
# short syntax
>>> yellow_fruits < yellow
True

>>> yellow.issuperset(yellow_fruits)
True
# short syntax
>>> yellow > yellow_fruits
True
```



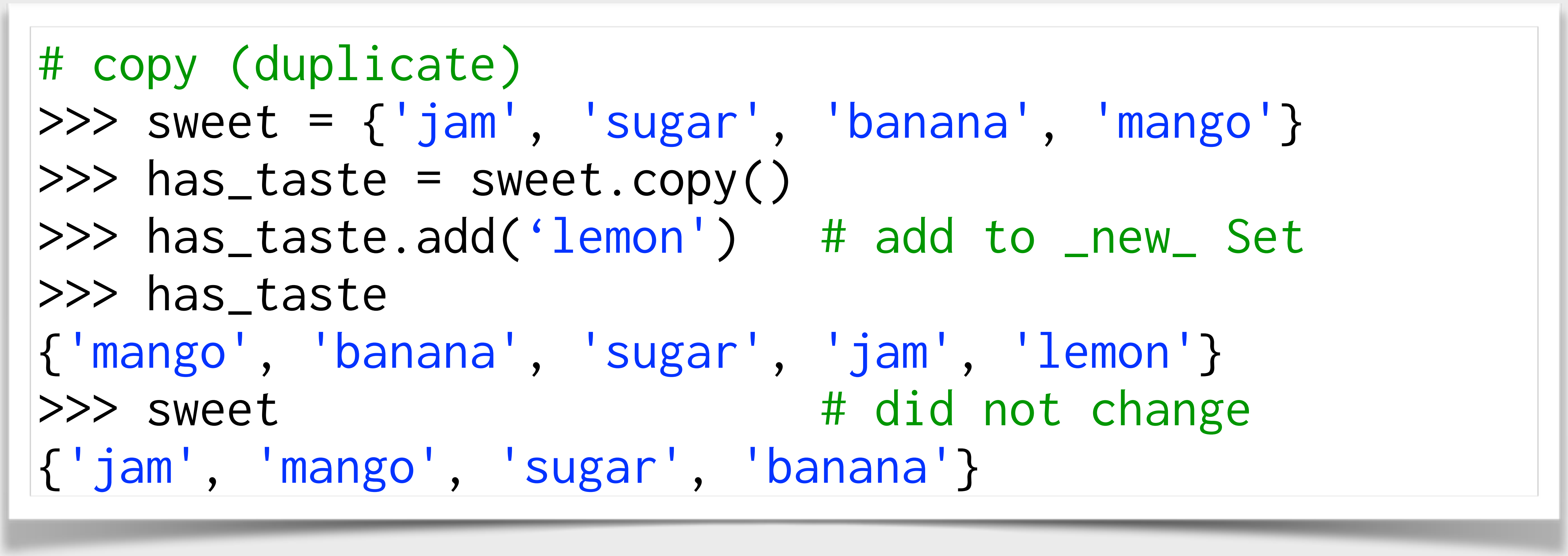
Disjoint



```
>>> sweet = {'jam', 'sugar',
             'banana', 'mango'}
>>> cars = {'Toyota', 'Mazda',
            'Ford'}
>>> sweet.isdisjoint(cars)
True
```

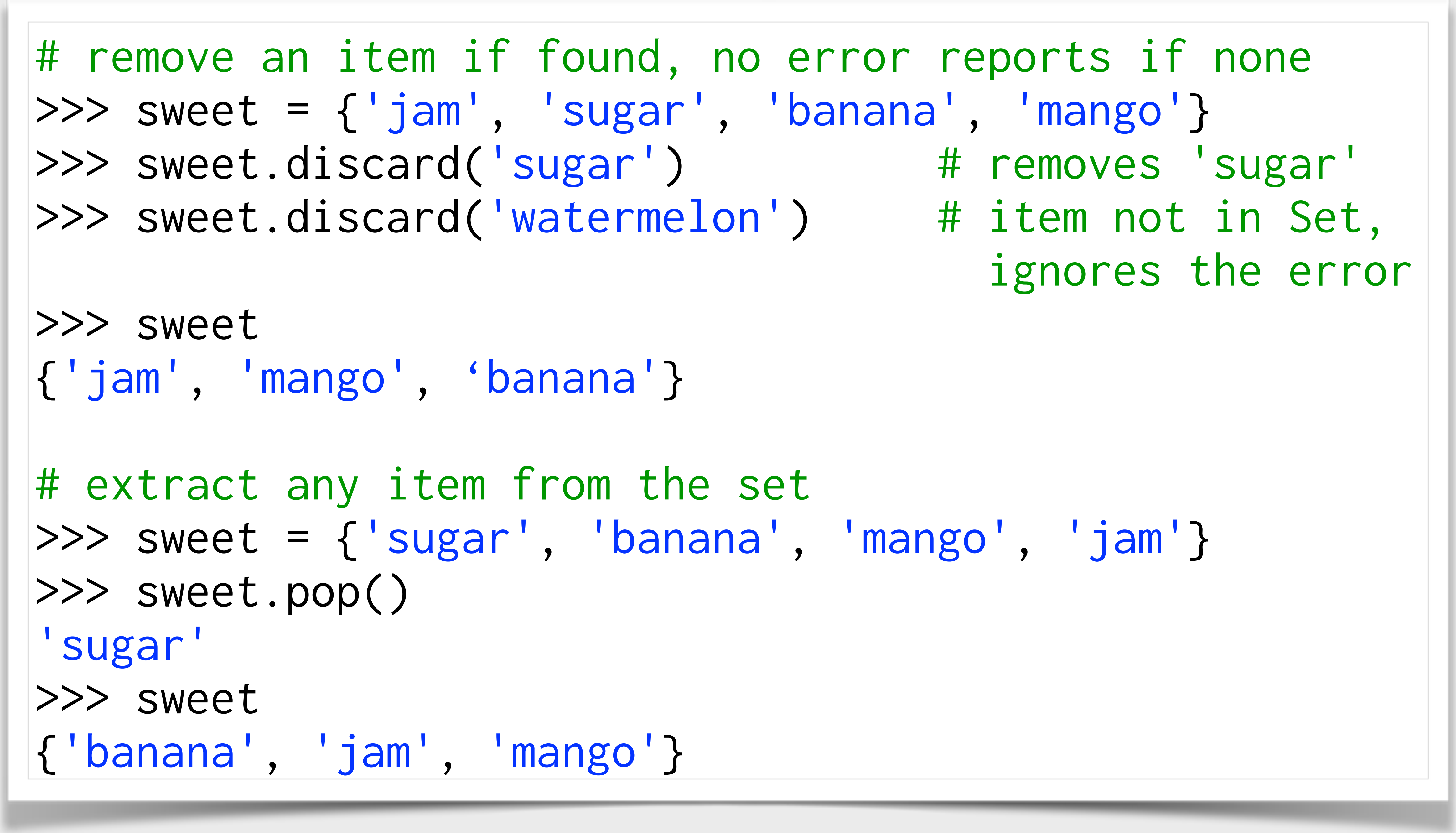


Set operations



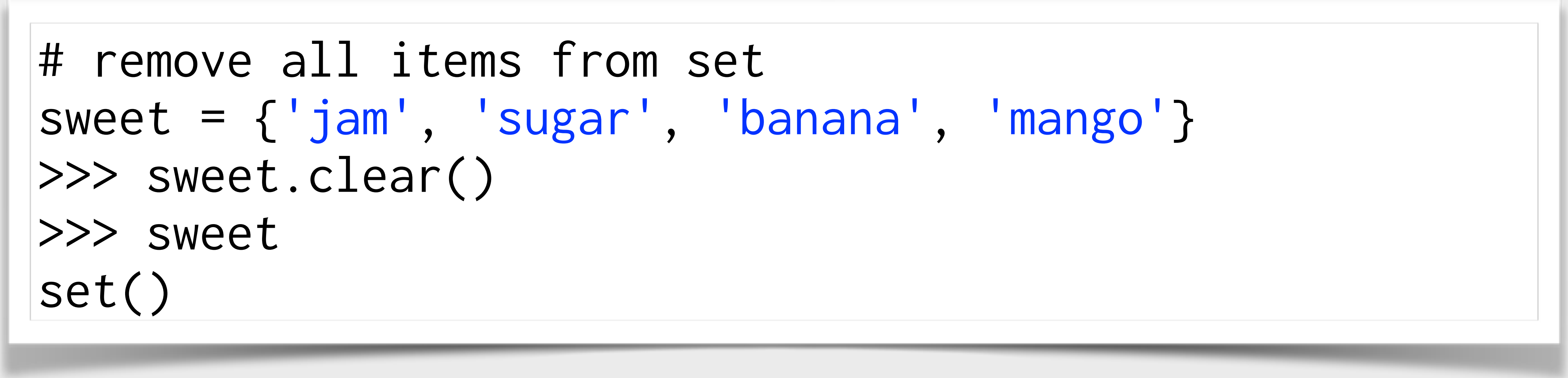
```
# copy (duplicate)
>>> sweet = {'jam', 'sugar', 'banana', 'mango'}
>>> has_taste = sweet.copy()
>>> has_taste.add(‘lemon')   # add to _new_ Set
>>> has_taste
{'mango', 'banana', 'sugar', 'jam', 'lemon'}
>>> sweet                    # did not change
{'jam', 'mango', 'sugar', 'banana'}
```

Set operations (2)



```
# remove an item if found, no error reports if none
>>> sweet = {'jam', 'sugar', 'banana', 'mango'}
>>> sweet.discard('sugar')          # removes 'sugar'
>>> sweet.discard('watermelon')     # item not in Set,
                                      ignores the error
>>> sweet
{'jam', 'mango', ‘banana'}

# extract any item from the set
>>> sweet = {'sugar', 'banana', 'mango', 'jam'}
>>> sweet.pop()
'sugar'
>>> sweet
{'banana', 'jam', 'mango'}
```

Set operations (3)



```
# remove all items from set
sweet = {'jam', 'sugar', 'banana', 'mango'}
>>> sweet.clear()
>>> sweet
set()
```

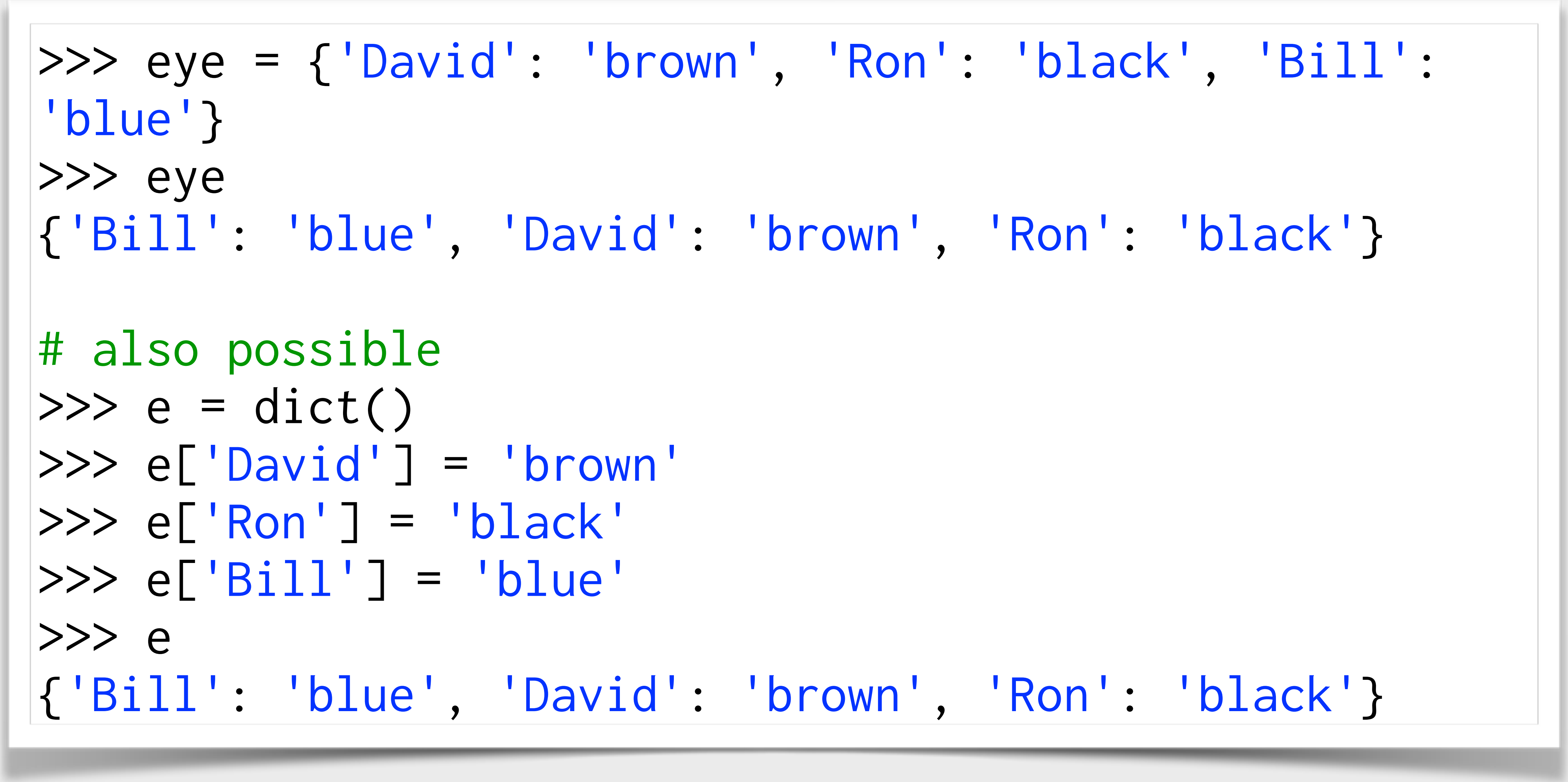
Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

Dictionary

- Key: Value mapping
- Item access in O(1)

Dictionary creation



```
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> eye
{'Bill': 'blue', 'David': 'brown', 'Ron': 'black'}

# also possible
>>> e = dict()
>>> e['David'] = 'brown'
>>> e['Ron'] = 'black'
>>> e['Bill'] = 'blue'
>>> e
{'Bill': 'blue', 'David': 'brown', 'Ron': 'black'}
```

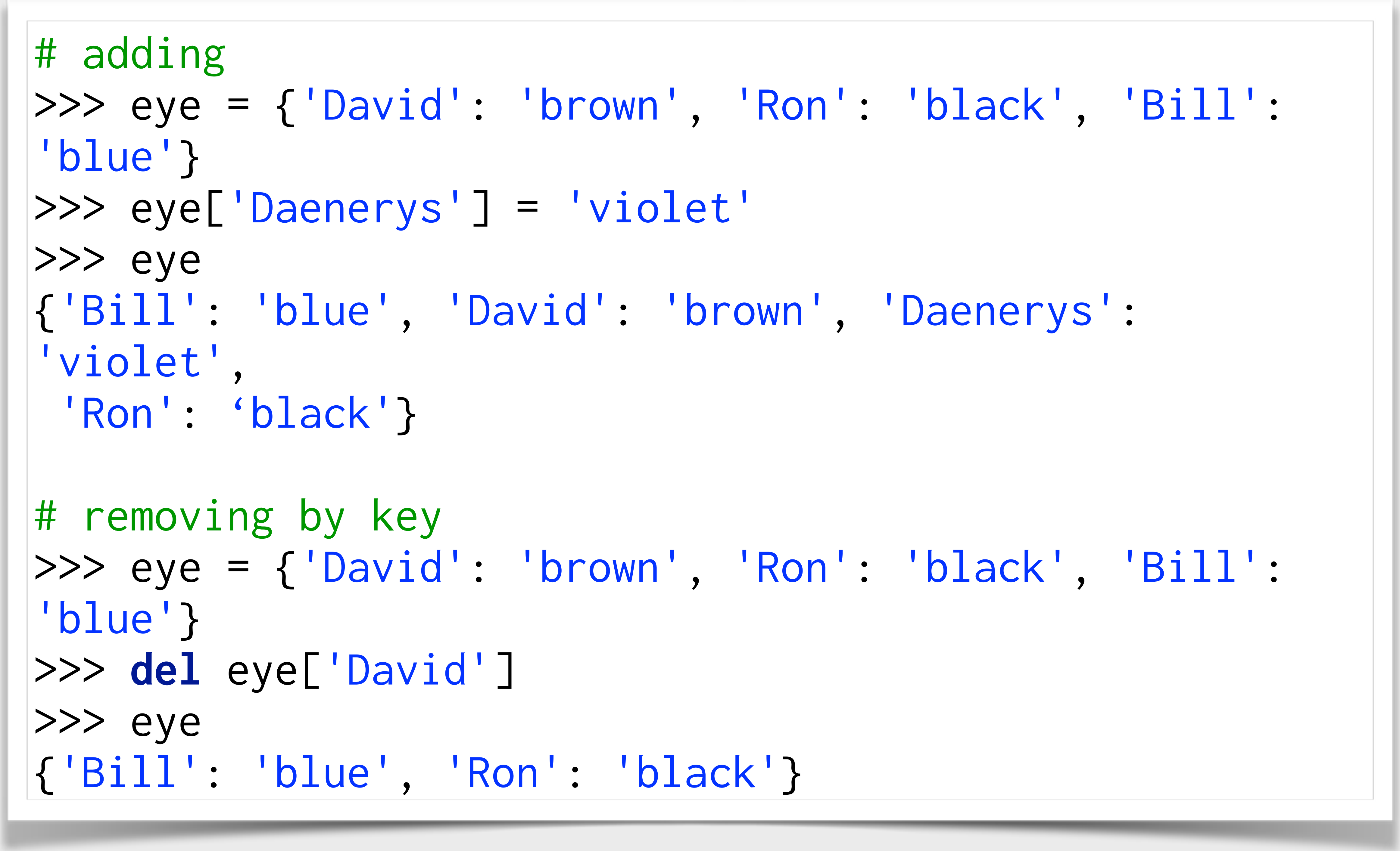
Keys - immutable

- number
- string
- tuple

Value - any type

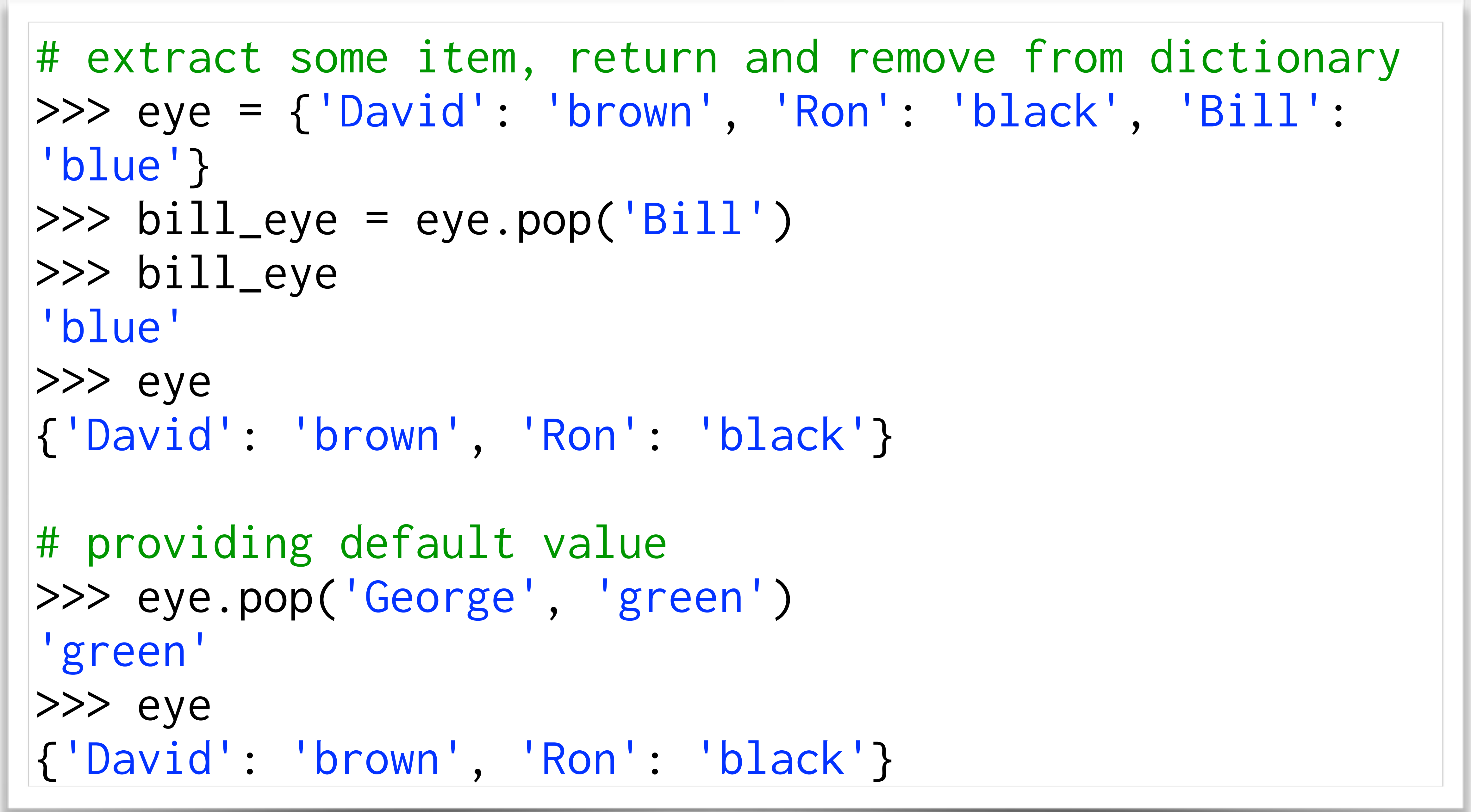
- immutable types
- lists
- sets
- dictionaries
- user-created types (we'll soon meet)

Dictionary actions



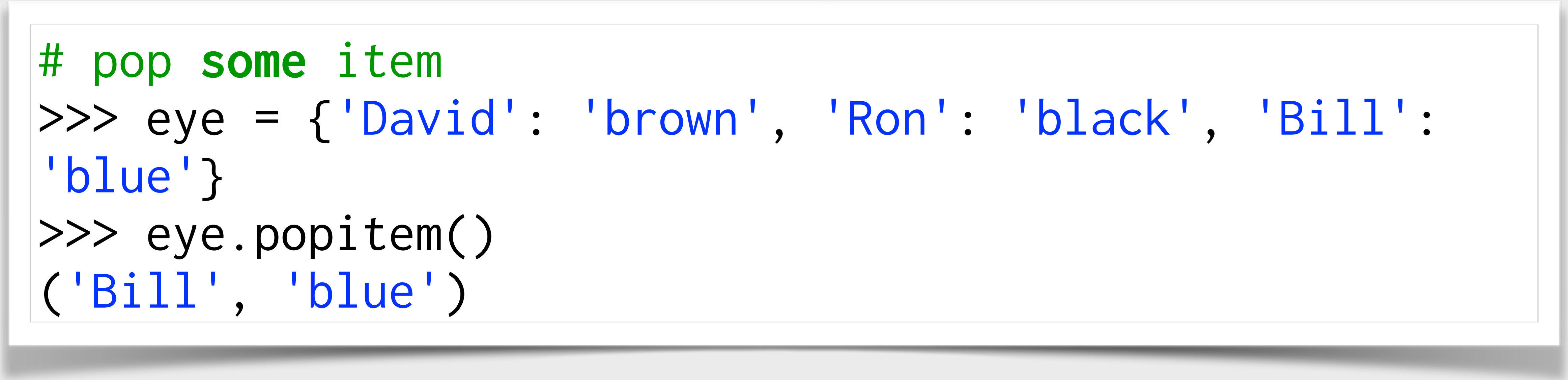
```
# adding
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> eye['Daenerys'] = 'violet'
>>> eye
{'Bill': 'blue', 'David': 'brown', 'Daenerys':
'violet',
 'Ron': ‘black'}

# removing by key
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> del eye['David']
>>> eye
{'Bill': 'blue', 'Ron': 'black'}
```

Dictionary actions



```
# extract some item, return and remove from dictionary
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> bill_eye = eye.pop('Bill')
>>> bill_eye
'blue'
>>> eye
{'David': 'brown', 'Ron': 'black'}

# providing default value
>>> eye.pop('George', 'green')
'green'
>>> eye
{'David': 'brown', 'Ron': 'black'}
```

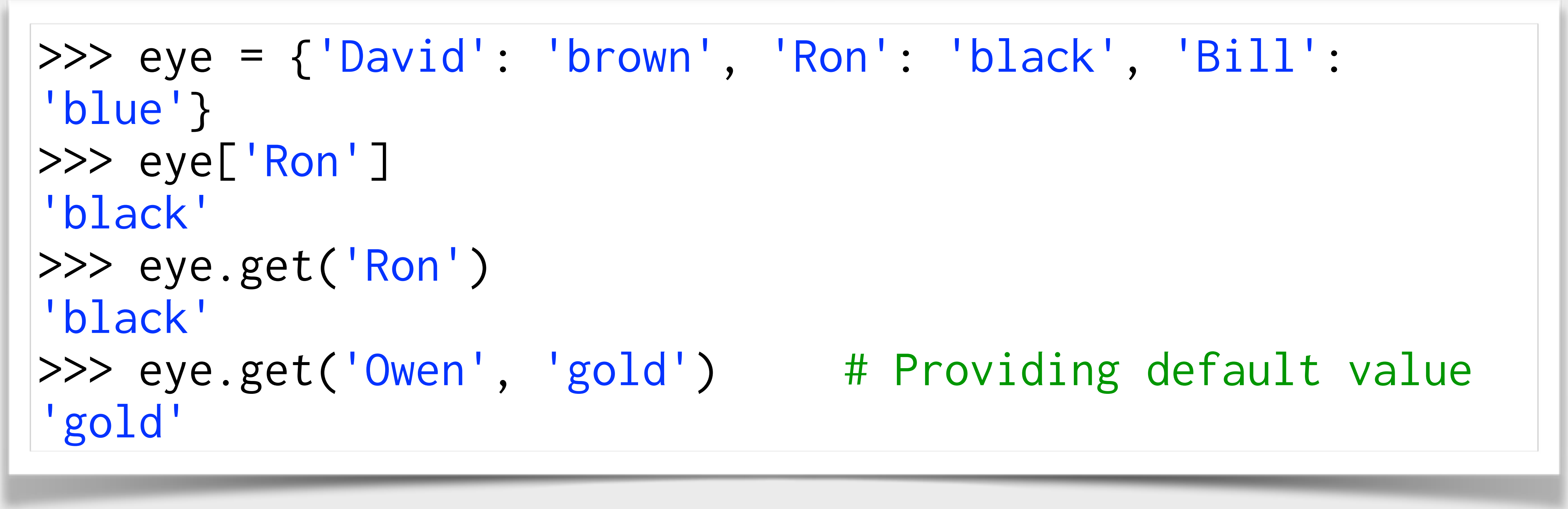
Dictionary actions (2)



```
# pop some item
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> eye.popitem()
('Bill', 'blue')
```

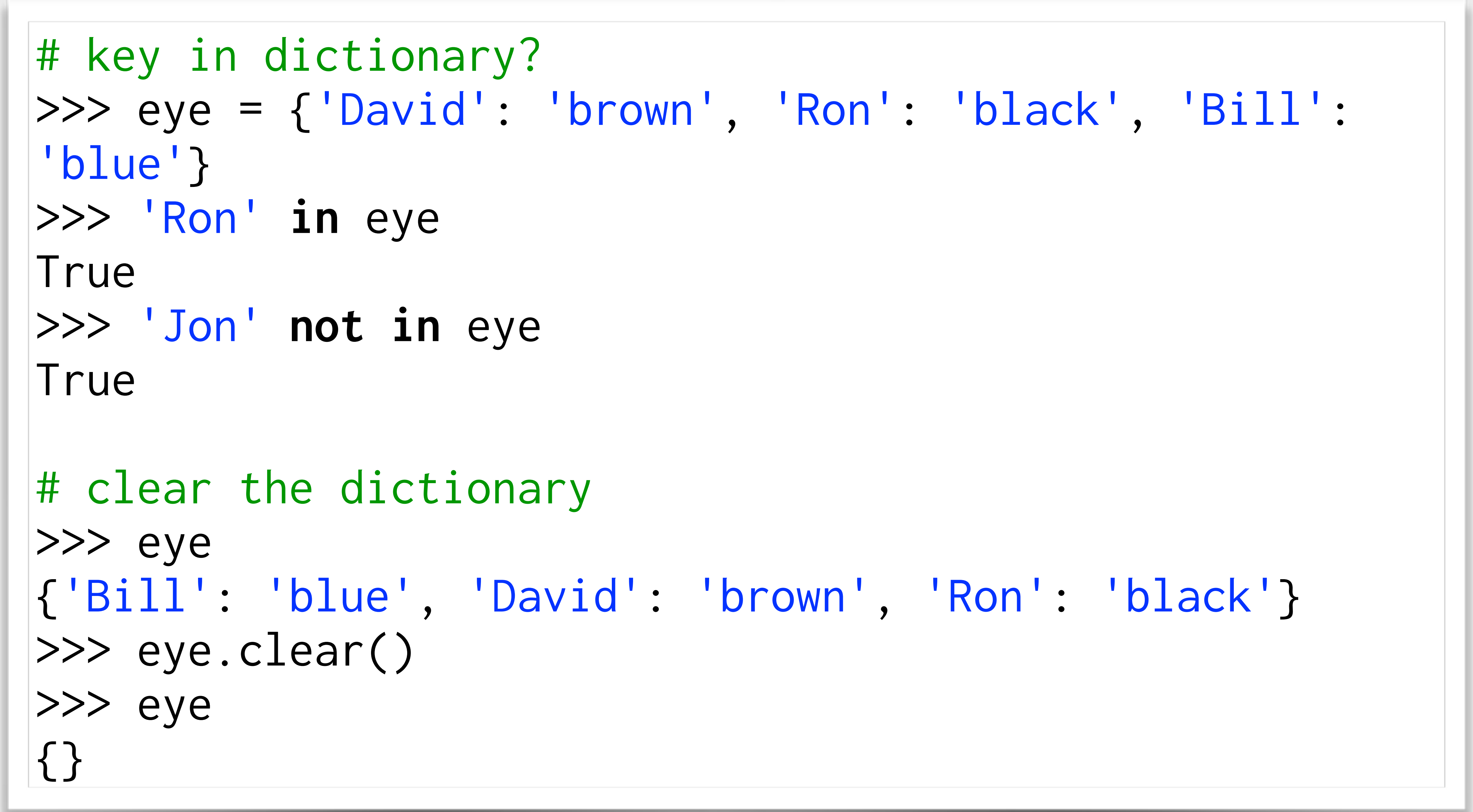
Dictionary: get value by key

Method	Returns
`d[key]`	value for `key`, error if does not exist
`get(key, default_val)`	value for `key` if exists, `default_val` otherwise



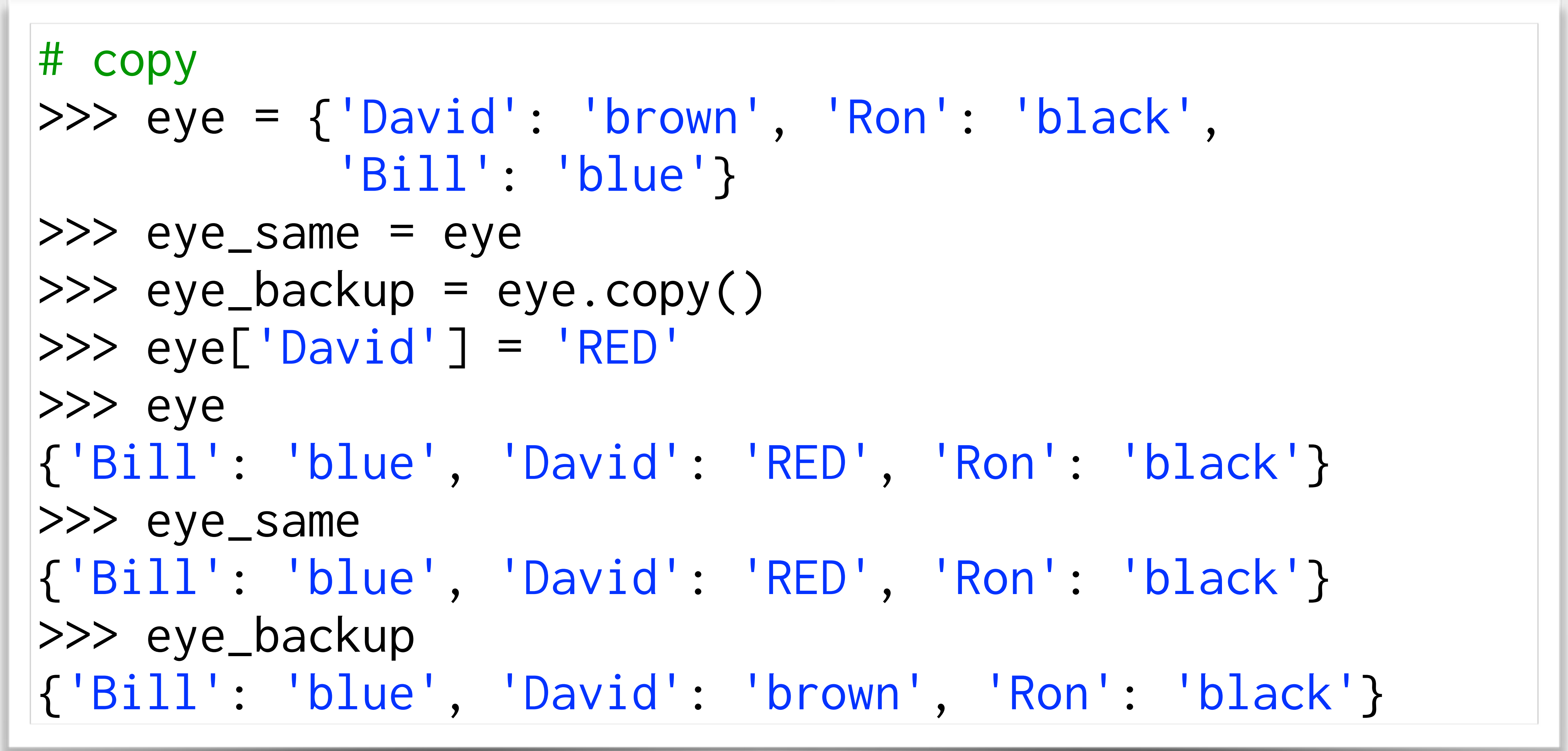
```
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> eye['Ron']
'black'
>>> eye.get('Ron')
'black'
>>> eye.get('Owen', 'gold')      # Providing default value
'gold'
```

Dictionary methods



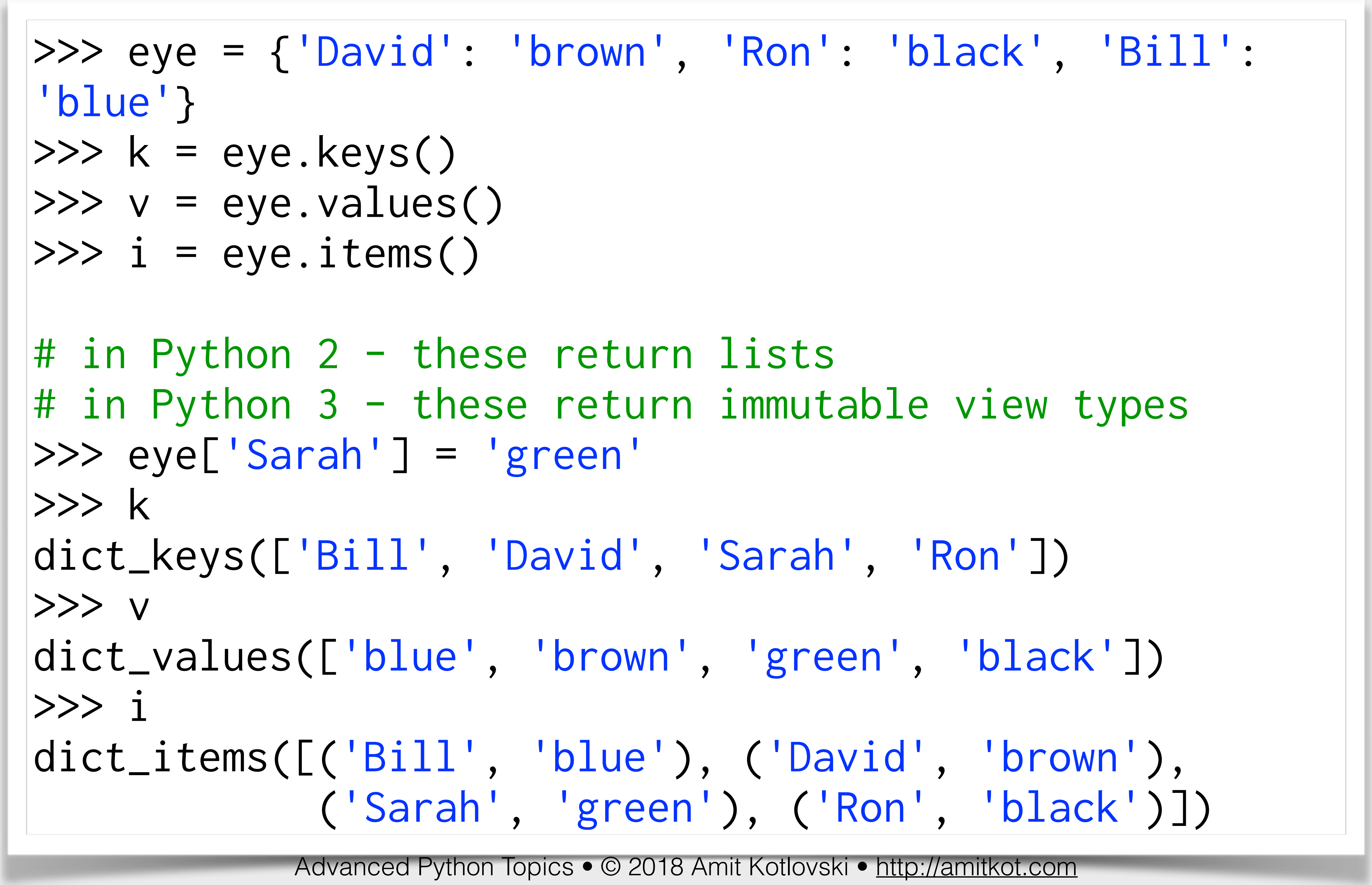
```
# key in dictionary?
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> 'Ron' in eye
True
>>> 'Jon' not in eye
True

# clear the dictionary
>>> eye
{'Bill': 'blue', 'David': 'brown', 'Ron': 'black'}
>>> eye.clear()
>>> eye
{}
```

Dictionary methods (2)



```
# copy
>>> eye = {'David': 'brown', 'Ron': 'black',
           'Bill': 'blue'}
>>> eye_same = eye
>>> eye_backup = eye.copy()
>>> eye['David'] = 'RED'
>>> eye
{'Bill': 'blue', 'David': 'RED', 'Ron': 'black'}
>>> eye_same
{'Bill': 'blue', 'David': 'RED', 'Ron': 'black'}
>>> eye_backup
{'Bill': 'blue', 'David': 'brown', 'Ron': 'black'}
```

keys(), values() and items()



```
>>> eye = {'David': 'brown', 'Ron': 'black', 'Bill':
'blue'}
>>> k = eye.keys()
>>> v = eye.values()
>>> i = eye.items()

# in Python 2 - these return lists
# in Python 3 - these return immutable view types
>>> eye['Sarah'] = 'green'
>>> k
dict_keys(['Bill', 'David', 'Sarah', 'Ron'])
>>> v
dict_values(['blue', 'brown', 'green', 'black'])
>>> i
dict_items([('Bill', 'blue'), ('David', 'brown'),
            ('Sarah', 'green'), ('Ron', 'black')])
```

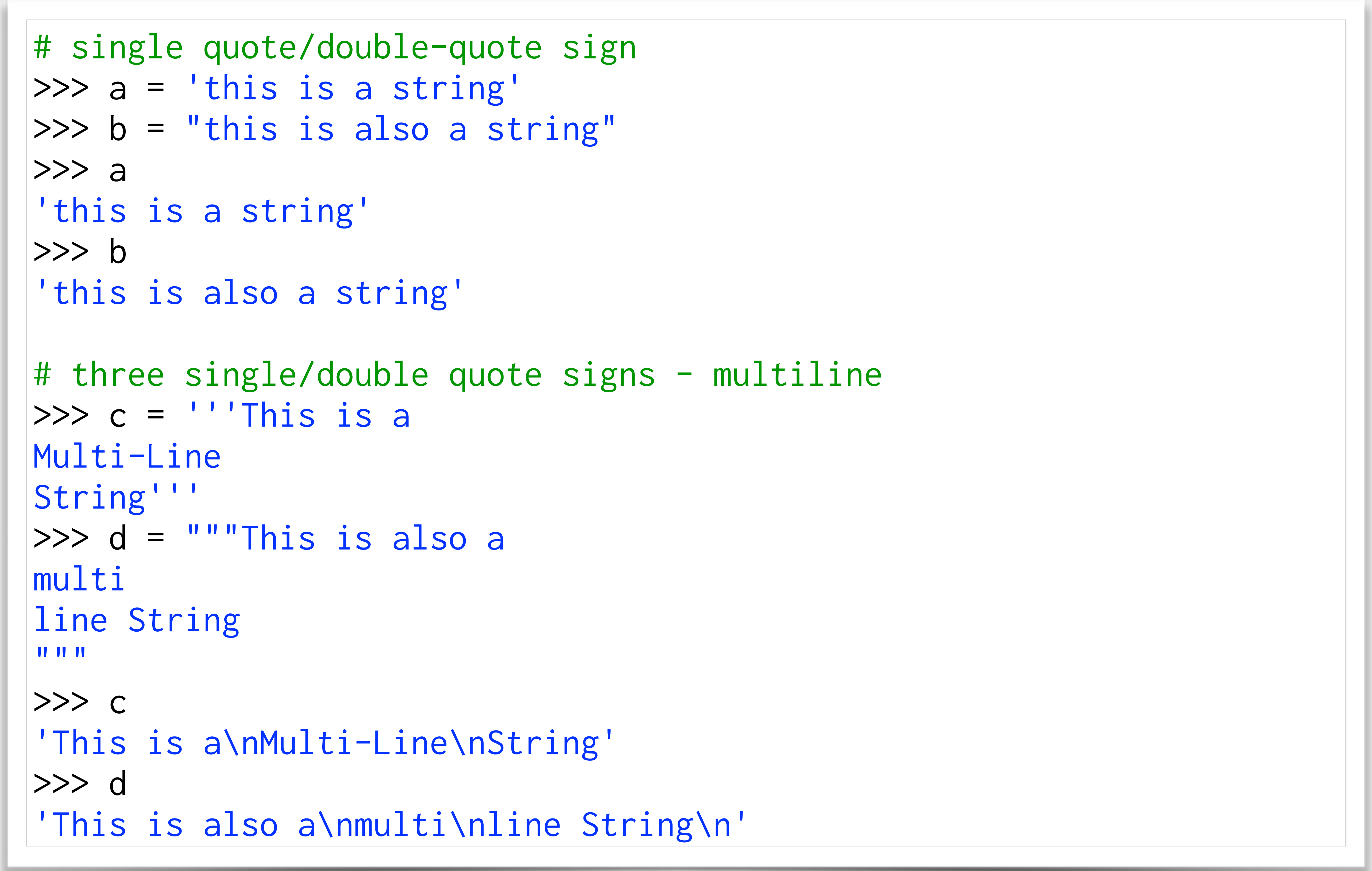
Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

string

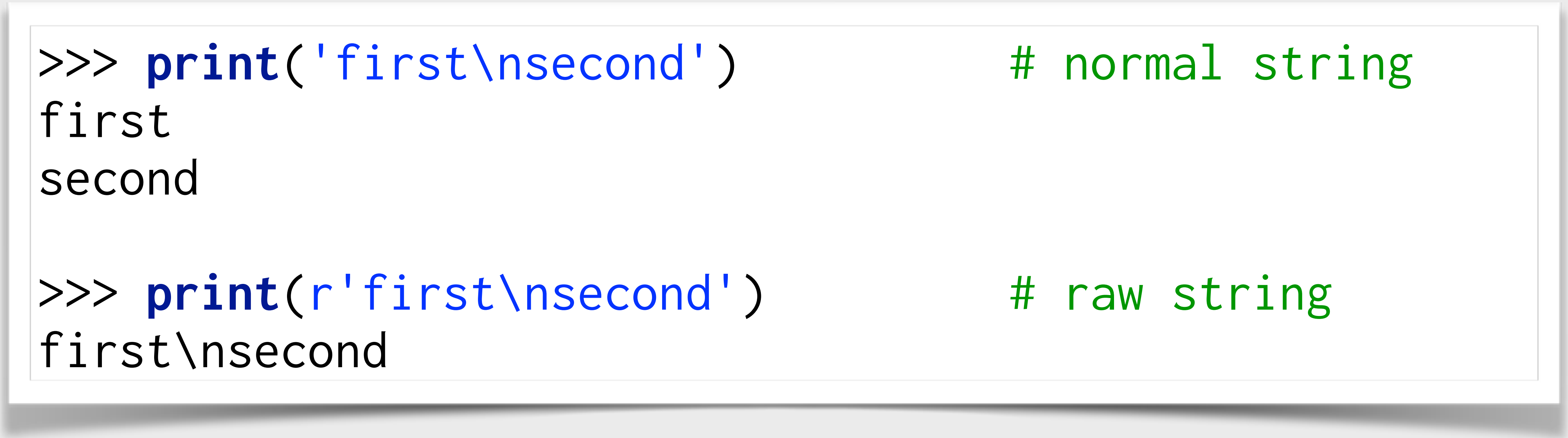
- An immutable sequence of characters

Building string



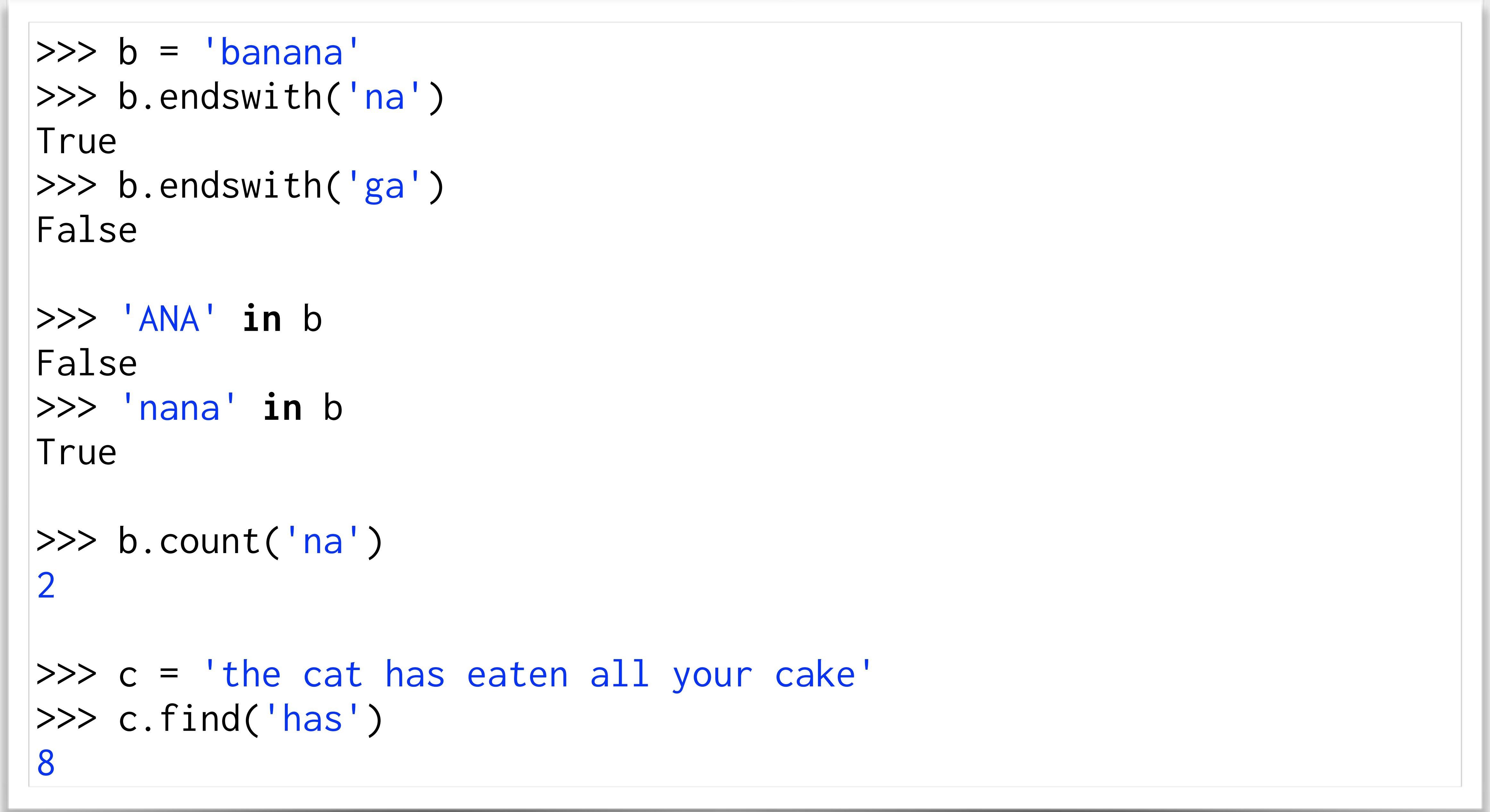
```
# single quote/double-quote sign
>>> a = 'this is a string'
>>> b = "this is also a string"
>>> a
'this is a string'
>>> b
'this is also a string'

# three single/double quote signs - multiline
>>> c = '''This is a
Multi-Line
String'''
>>> d = """This is also a
multi
line String
"""
>>> c
'This is a\nMulti-Line\nString'
>>> d
'This is also a\nmulti\nline String\n'
```

Raw string



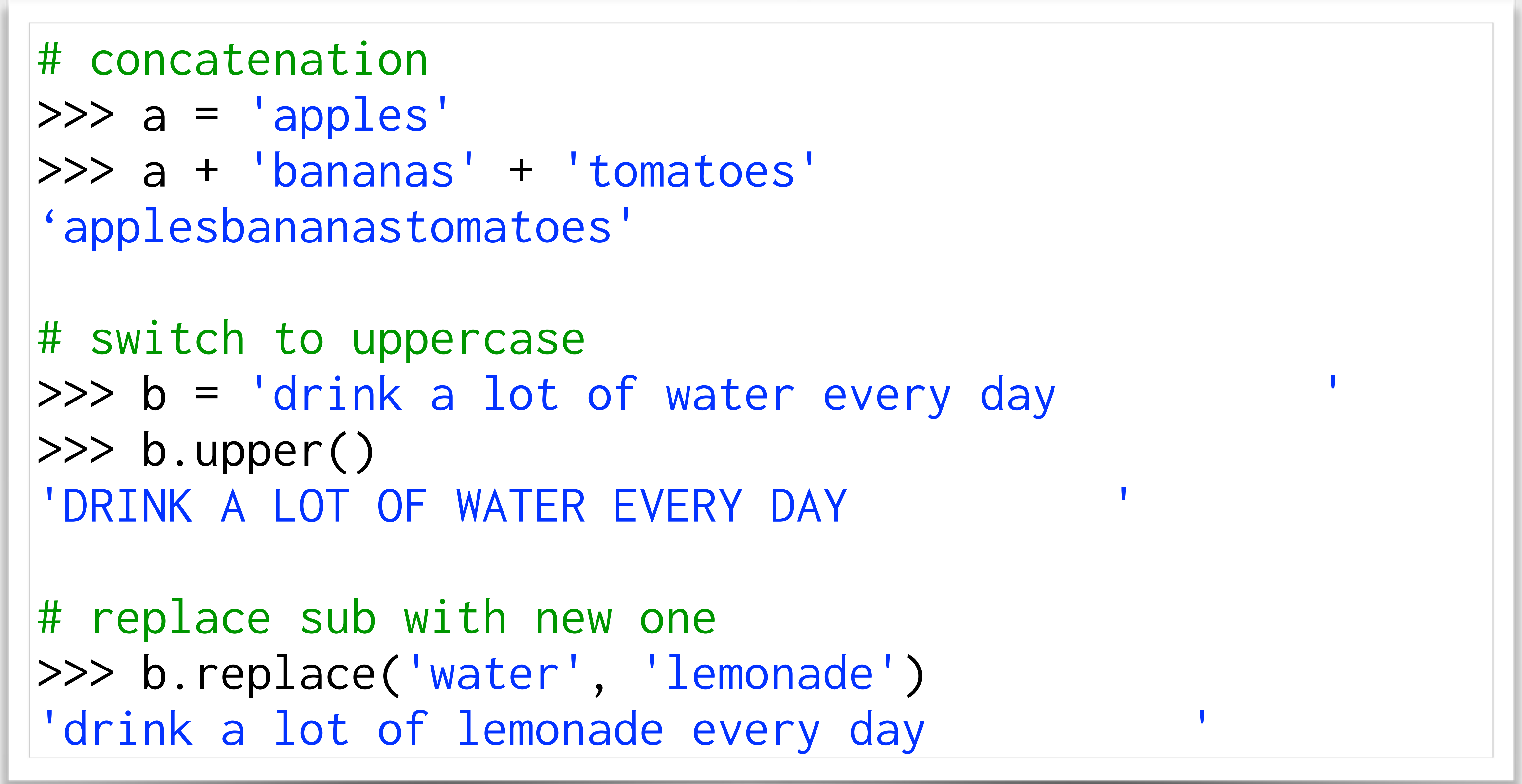
```
>>> print('first\nsecond')               # normal string
first
second

>>> print(r'first\nsecond')              # raw string
first\nsecond
```

String testing



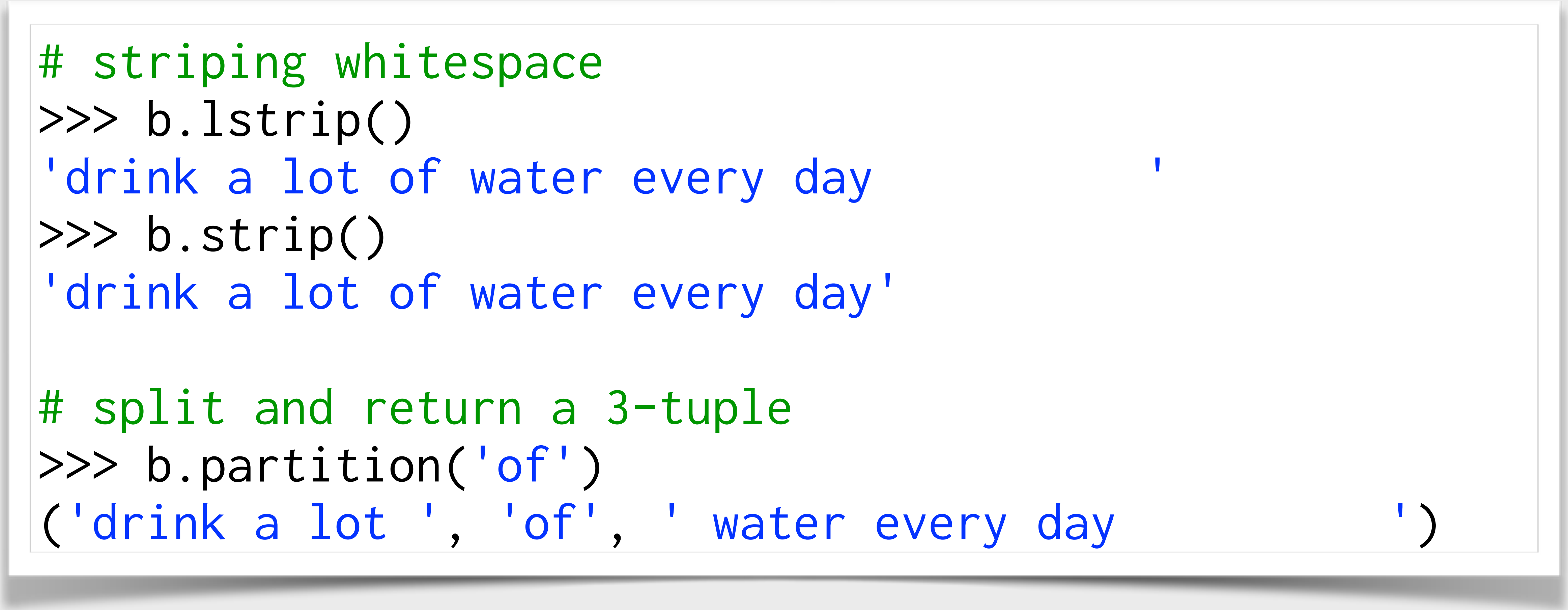
```
>>> b = 'banana'
>>> b.endswith('na')
True
>>> b.endswith('ga')
False

>>> 'ANA' in b
False
>>> 'nana' in b
True

>>> b.count('na')
2

>>> c = 'the cat has eaten all your cake'
>>> c.find('has')
8
```

String operations



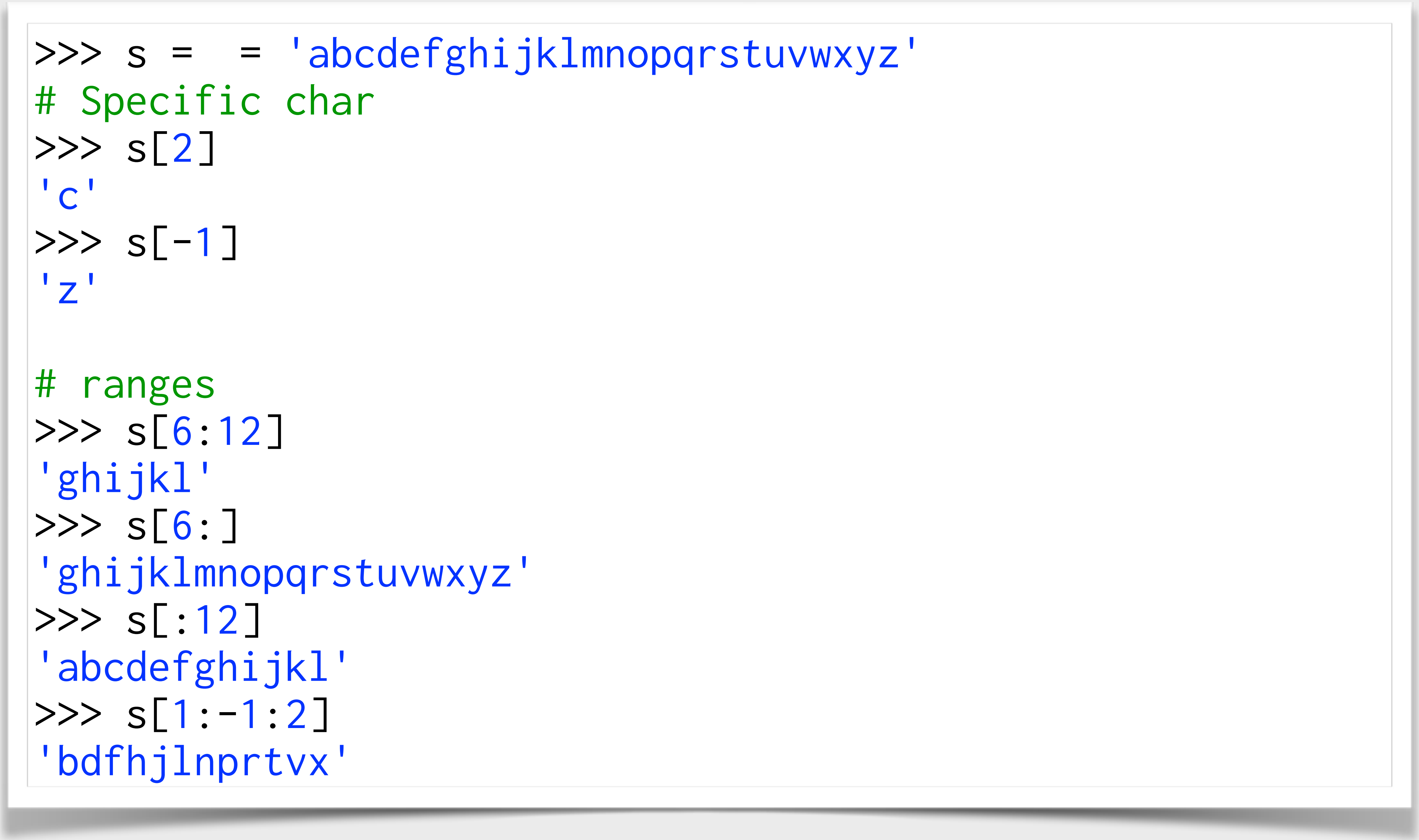
```
# concatenation
>>> a = 'apples'
>>> a + 'bananas' + 'tomatoes'
‘applesbananastomatoes'

# switch to uppercase
>>> b = 'drink a lot of water every day          '
>>> b.upper()
'DRINK A LOT OF WATER EVERY DAY          '

# replace sub with new one
>>> b.replace('water', 'lemonade')
'drink a lot of lemonade every day          '
```

String operations (2)



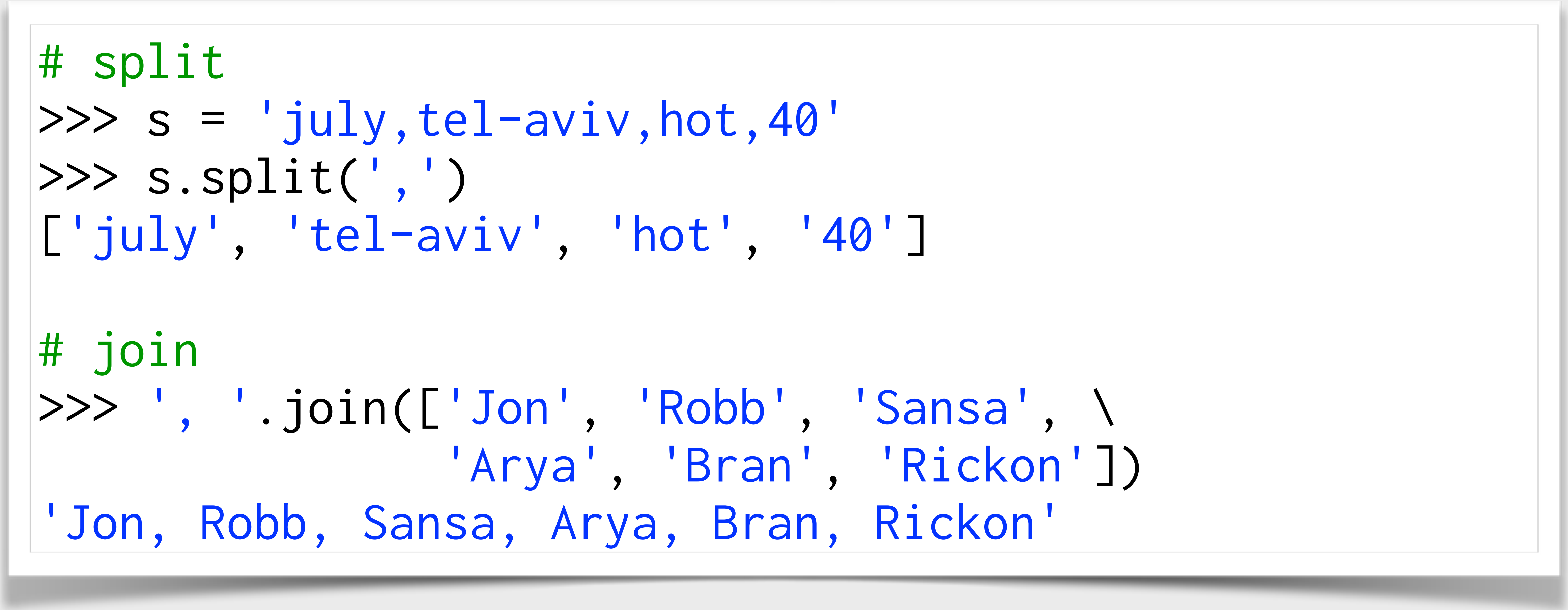
```
# striping whitespace
>>> b.lstrip()
'drink a lot of water every day          '
>>> b.strip()
'drink a lot of water every day'

# split and return a 3-tuple
>>> b.partition('of')
('drink a lot ', 'of', ' water every day          ')
```

Slice



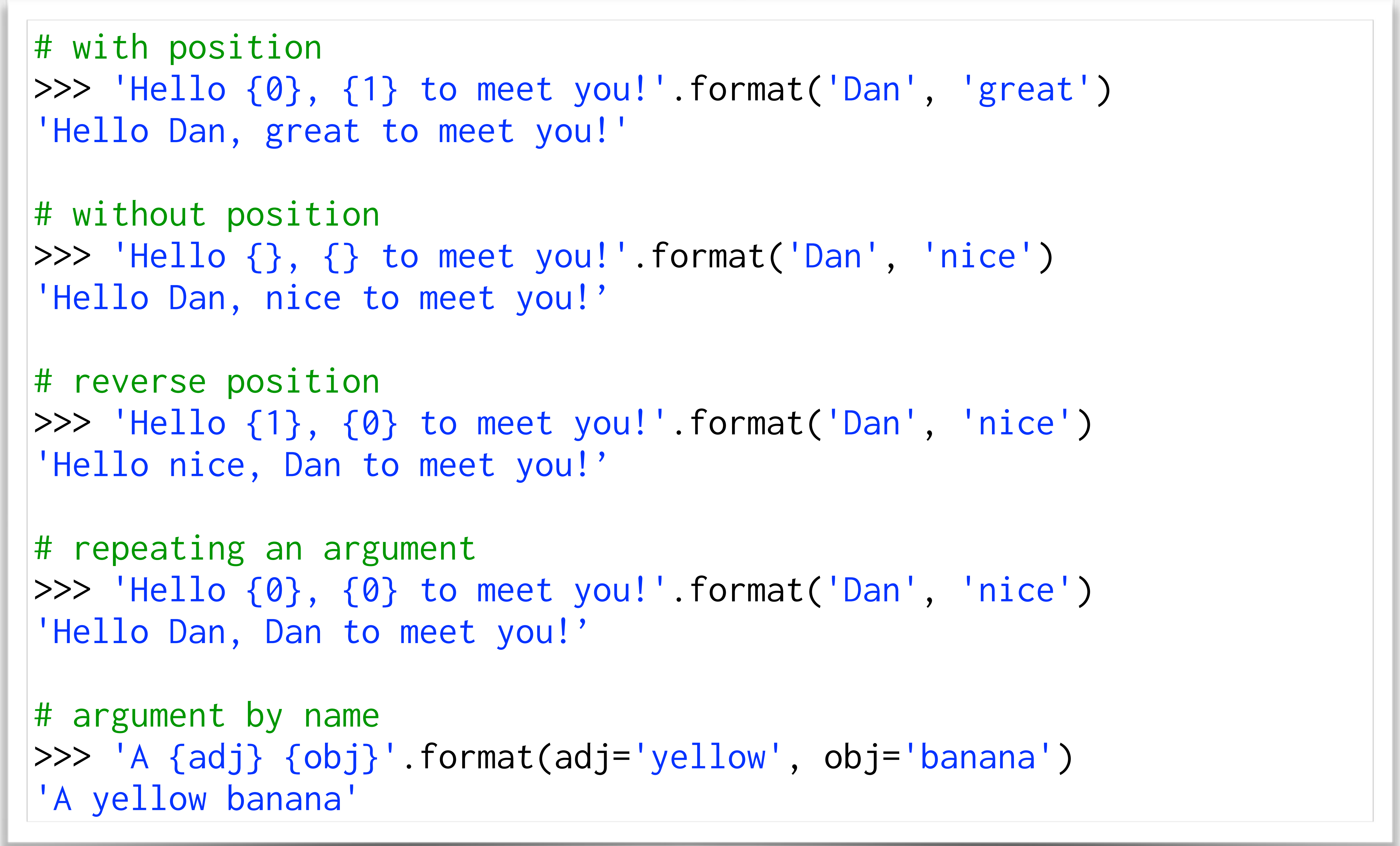
```
>>> s =  = 'abcdefghijklmnopqrstuvwxyz'
# Specific char
>>> s[2]
'c'
>>> s[-1]
'z'

# ranges
>>> s[6:12]
'ghijkl'
>>> s[6:]
'ghijklmnopqrstuvwxyz'
>>> s[:12]
'abcdefghijkl'
>>> s[1:-1:2]
'bdfhjlnprtvx'
```

split and join



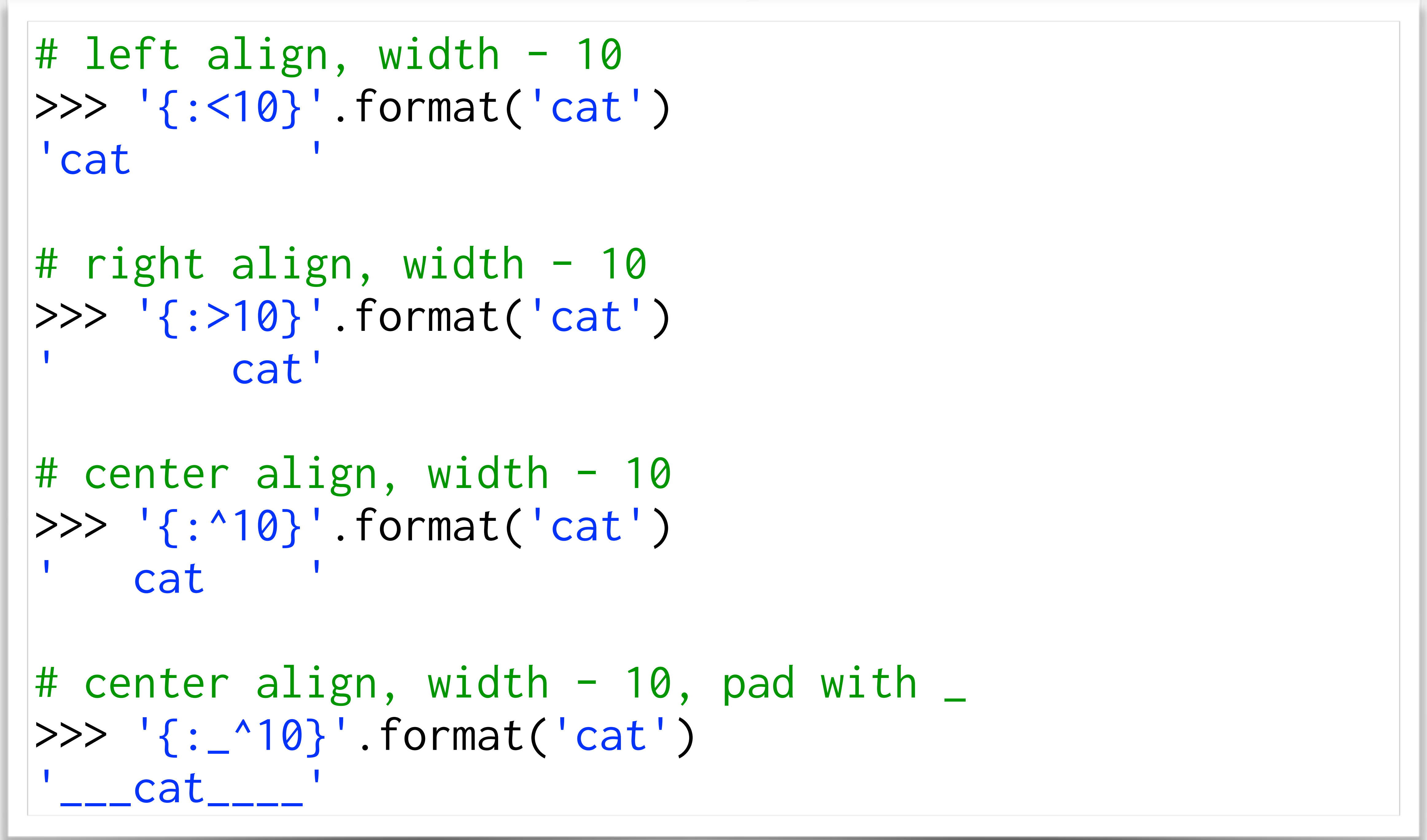
```
# split
>>> s = 'july,tel-aviv,hot,40'
>>> s.split(',')
['july', 'tel-aviv', 'hot', '40']

# join
>>> ', '.join(['Jon', 'Robb', 'Sansa', \
               'Arya', 'Bran', 'Rickon'])
'Jon, Robb, Sansa, Arya, Bran, Rickon'
```

String format



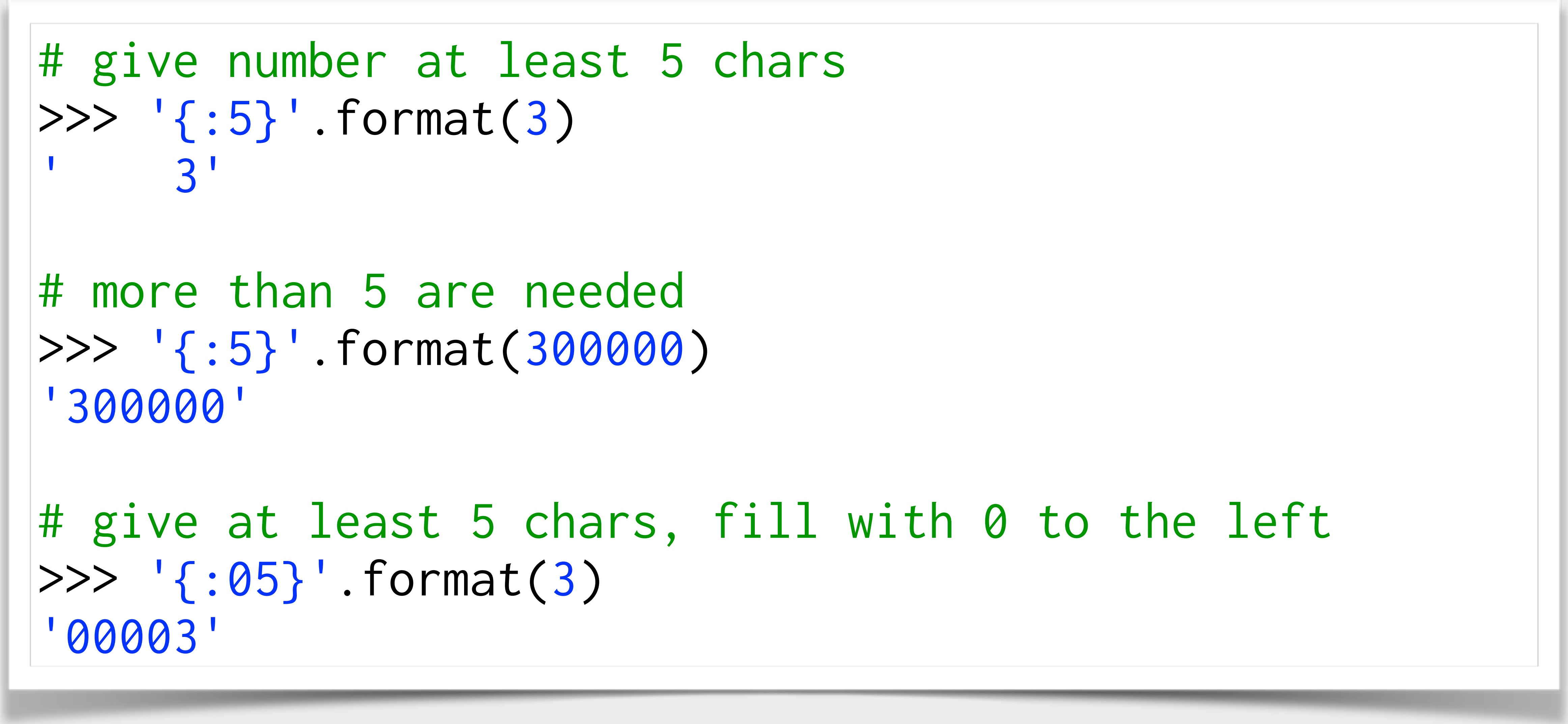
```
# with position
>>> 'Hello {0}, {1} to meet you!'.format('Dan', 'great')
'Hello Dan, great to meet you!'

# without position
>>> 'Hello {}, {} to meet you!'.format('Dan', 'nice')
'Hello Dan, nice to meet you!’

# reverse position
>>> 'Hello {1}, {0} to meet you!'.format('Dan', 'nice')
'Hello nice, Dan to meet you!’

# repeating an argument
>>> 'Hello {0}, {0} to meet you!'.format('Dan', 'nice')
'Hello Dan, Dan to meet you!’

# argument by name
>>> 'A {adj} {obj}'.format(adj='yellow', obj='banana')
'A yellow banana'
```

String format (2)



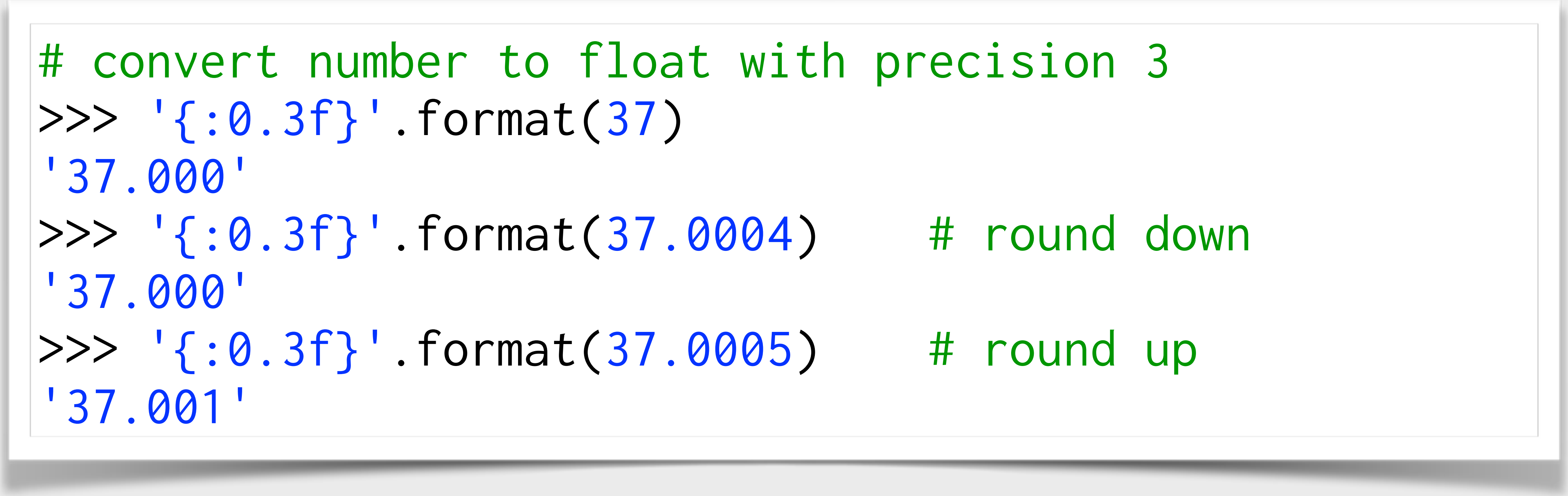
```
# left align, width - 10
>>> '{:<10}'.format('cat')
'cat       '

# right align, width - 10
>>> '{:>10}'.format('cat')
'       cat'

# center align, width - 10
>>> '{:^10}'.format('cat')
'   cat    '

# center align, width - 10, pad with _
>>> '{:_^10}'.format('cat')
'___cat____'
```

String format (3)



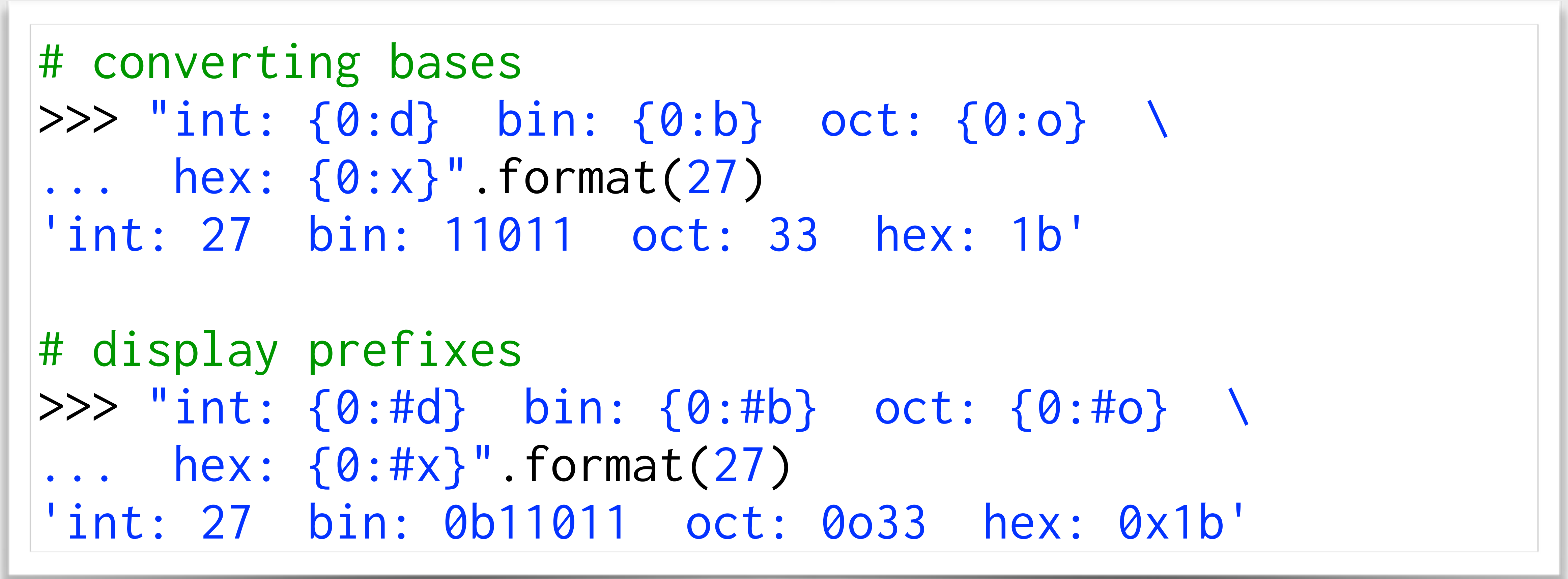
```
# give number at least 5 chars
>>> '{:5}'.format(3)
'    3'

# more than 5 are needed
>>> '{:5}'.format(300000)
'300000'

# give at least 5 chars, fill with 0 to the left
>>> '{:05}'.format(3)
'00003'
```

String format (4)



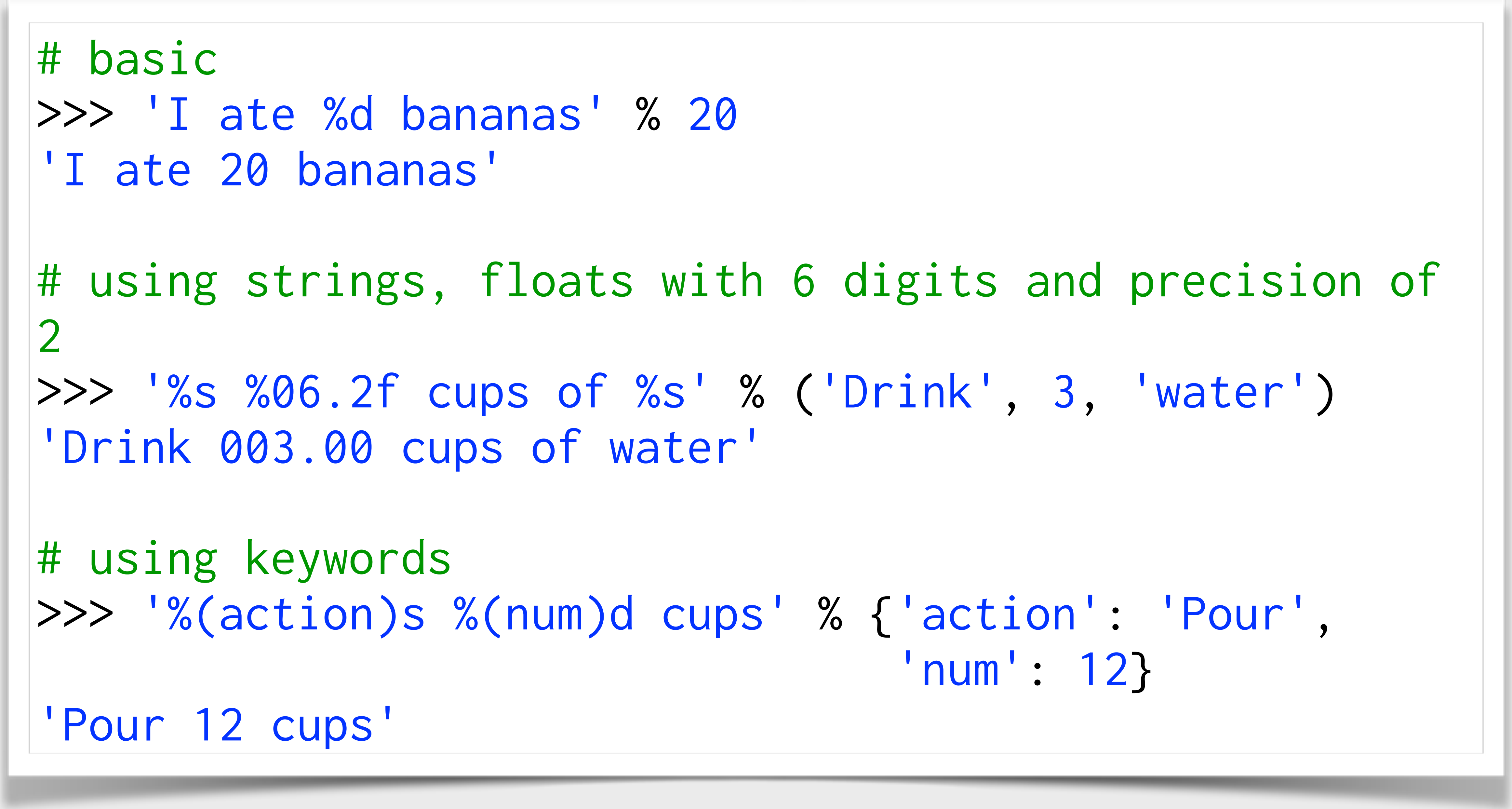
```
# convert number to float with precision 3
>>> '{:0.3f}'.format(37)
'37.000'
>>> '{:0.3f}'.format(37.0004)    # round down
'37.000'
>>> '{:0.3f}'.format(37.0005)    # round up
'37.001'
```

String format (5)



```
# converting bases
>>> "int: {0:d}  bin: {0:b}  oct: {0:o}  \
...   hex: {0:x}".format(27)
'int: 27  bin: 11011  oct: 33  hex: 1b'

# display prefixes
>>> "int: {0:#d}  bin: {0:#b}  oct: {0:#o}  \
...   hex: {0:#x}".format(27)
'int: 27  bin: 0b11011  oct: 0o33  hex: 0x1b'
```

Old C-style formatting



```
# basic
>>> 'I ate %d bananas' % 20
'I ate 20 bananas'

# using strings, floats with 6 digits and precision of
2
>>> '%s %06.2f cups of %s' % ('Drink', 3, 'water')
'Drink 003.00 cups of water'

# using keywords
>>> '%(action)s %(num)d cups' % {'action': 'Pour',
                                 'num': 12}
'Pour 12 cups'
```

Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

Summary

- Tuples are immutable lists
- Lists are ordered data collections with sort, find and more
- Sets are collections with no duplicate items
- Dictionaries map keys to values
- Strings are immutable sequences of characters

AC..
TEAS EF
TLW
QUESTION
EVERYTHING

Thanks!

twitter	@amitkot
www	amitkot.com