Python CS-521

Eugene Pinsky
Department of Computer Science
Metropolitan College, Boston University
Boston, MA 02215
email: epinsky@bu.edu

May 12, 2020

Abstract

This course will present an effective approach to help you learn Python. With extensive use of graphical illustrations, we will build understanding of Python and its capabilities by learning through many simple examples and analogies. The class will involve active student participation, discussions, and programming exercises. This approach will help you build a strong foundation in Python that you will be able to effectively apply in real-job situations and future courses.

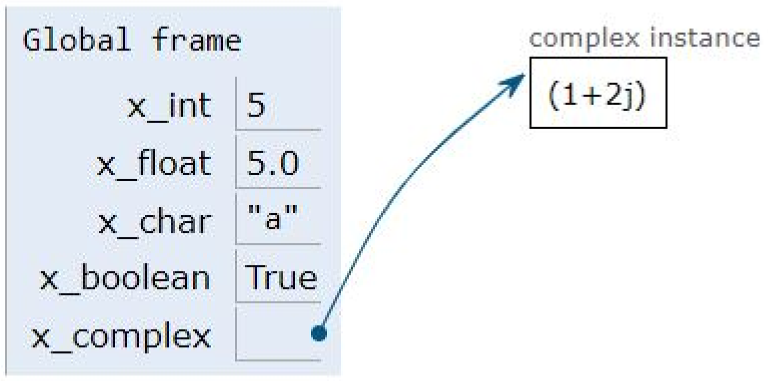
COLLECTIONS

Python Data Types

- data types define basic building blocks in a language
- similar to noun, verb
- Python has two groups of types
  1. primitive types (”atoms”)
  2. collections (”molecules”)
- additional special types:
  1. *None* type
  2. *range* type

Primitive Types

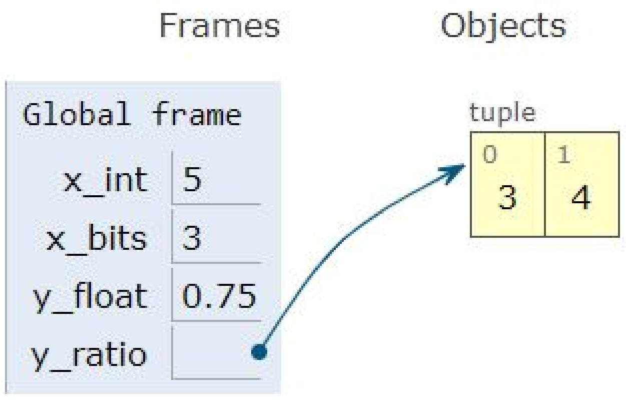
```
x_int     = 5
x_float   = 5.0
x_char    = 'a'
x_boolean = True
x_complex = 1 + 2j
```



- *"atoms"* - indivisible objects

Primitive Type Method Examples

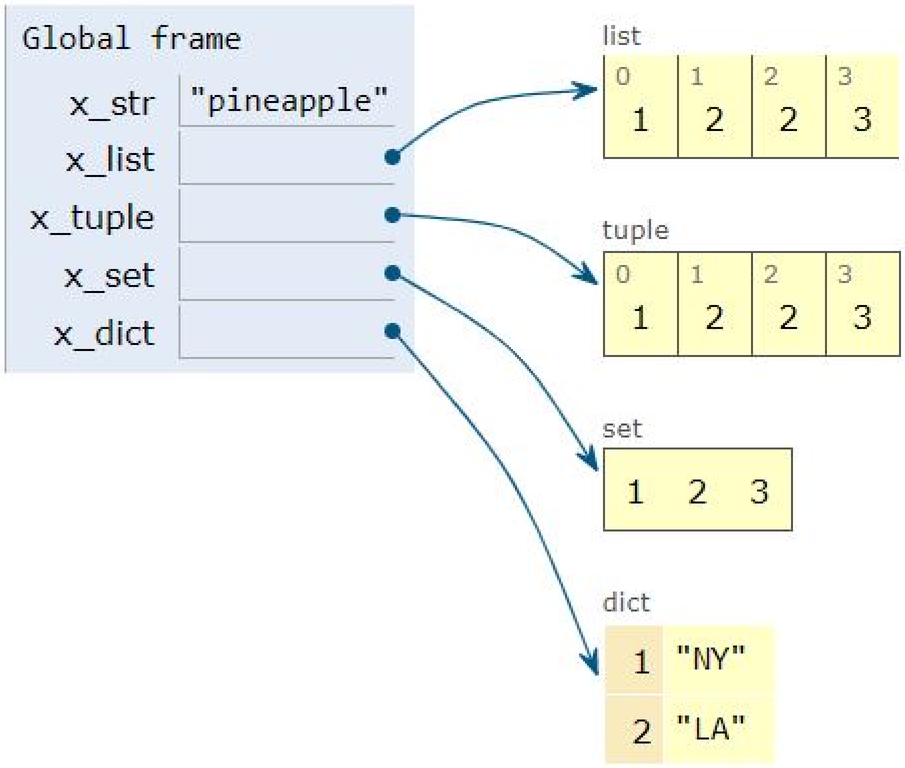
```
x_int   = 5
x_bits  = x_int.bit_length()

y_float = 0.75
y_ratio = y_float.as_integer_ratio()
```



- *"atoms"* are not just values
- objects with methods

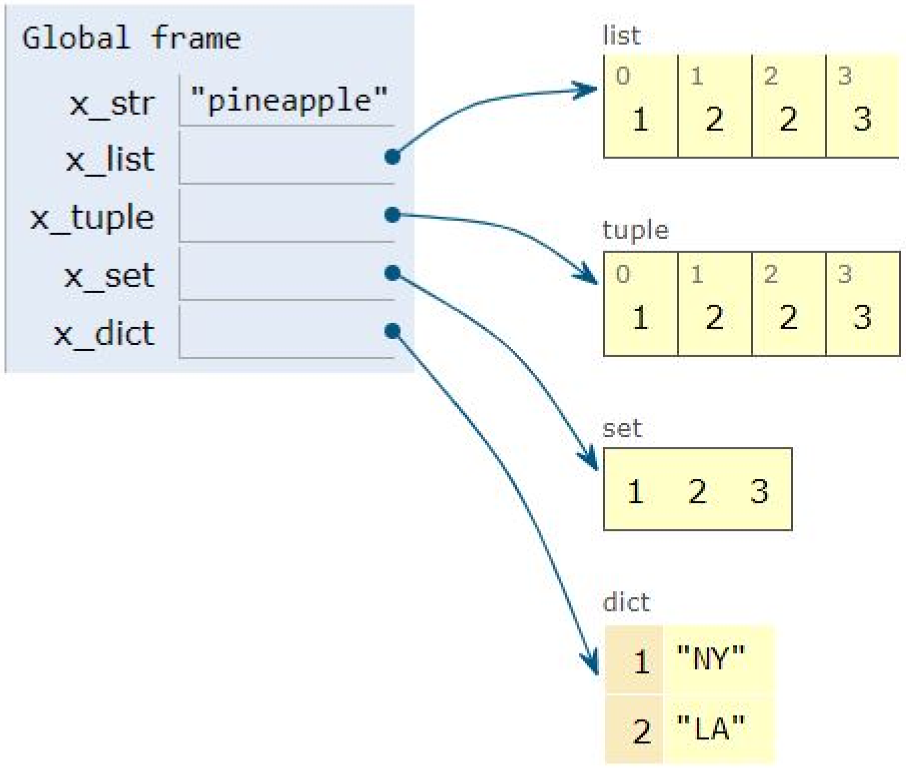
Collection Types

```
x_str   = 'pineapple'
x_list  = [1,  2, 2, 3]
x_tuple = (1, 2, 2, 3)
x_set   = {1, 2, 2, 3} # note duplicates
x_dict  = {1: 'NY', 2: 'LA'}
```



- *molecules*: complex objects

Collections



- membership: *in/not in*
- iteration: *for*

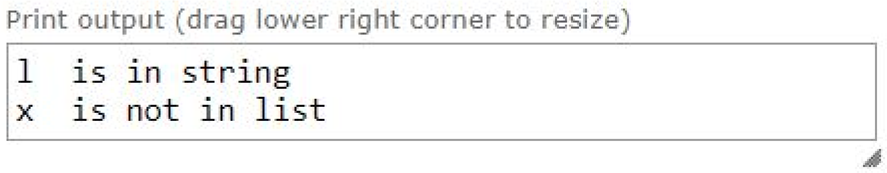
Exercise(s):

- For each line indicate object type (primitive or collection)

```
j = 5
y = 'a'
a = 2 + 2j
b = 2 + 2*j
c = [2, 2*j]
d = {j : a, y: b}
e = {j}
f = (j)
g = (j, )
```

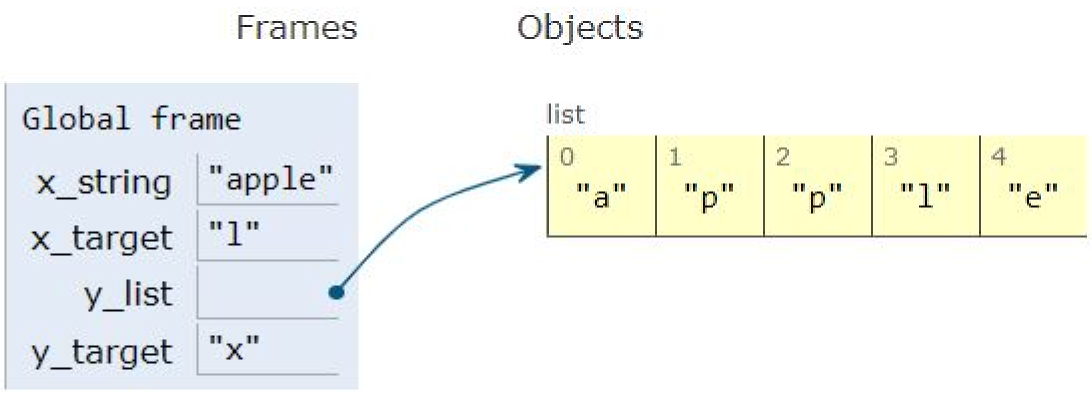
Membership: *in/not in*

```
x_string = 'apple'
x_target = 'l'
if x_target in x_string:
    print(x_target, ' is in string')

y_list   = ['a','p','p','l','e']
y_target = 'x'
if y_target not in y_list:
    print(y_target, ' is not in list')
```



Print output (drag lower right corner to resize)

```
l  is in string
x  is not in list
```



Exercise(s):

- construct three different collections containing words from *x_str*:

```
a cube has many symmetries
```

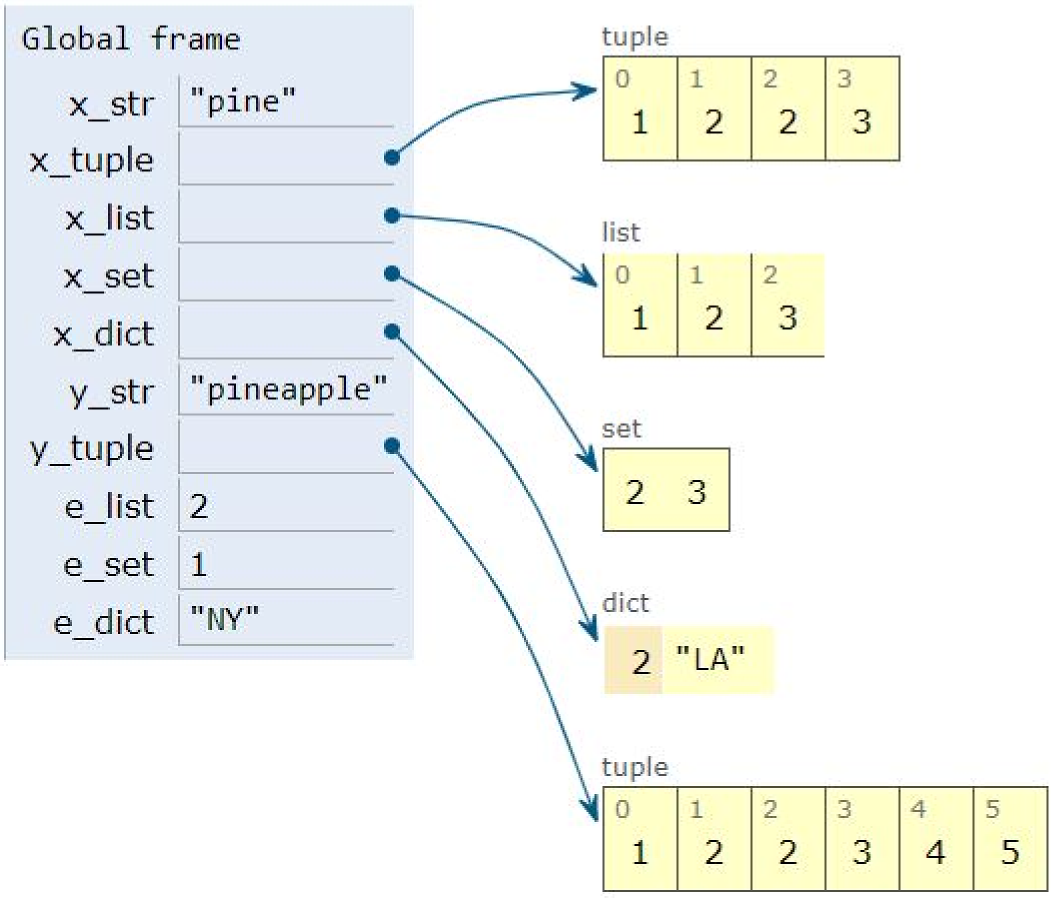
- for each collection use iteration to check if it contains the word *"many"*
- for each collection use *in, not in* check if it contains the word *"cube"*

Collection Methods

```
x_str   = 'pine'
x_tuple = (1, 2, 2, 3)
x_list  = [1, 2, 2, 3]
x_set   = {1, 2, 2, 3}
x_dict  = {1: 'NY', 2: 'LA'}

y_str   = x_str + 'apple'
y_tuple = x_tuple  + (4, 5)
e_list  = x_list.pop(1)
e_set   = x_set.pop()
e_dict  = x_dict.pop(1)
```

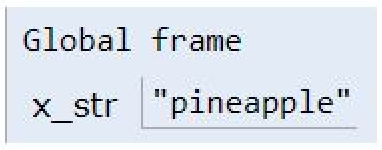
- '+' is overloaded
- *polymorphic* methods:
  same function, different types

Collection Methods (cont'd)



A Python String

```
x_str = 'pineapple'
```



Global frame

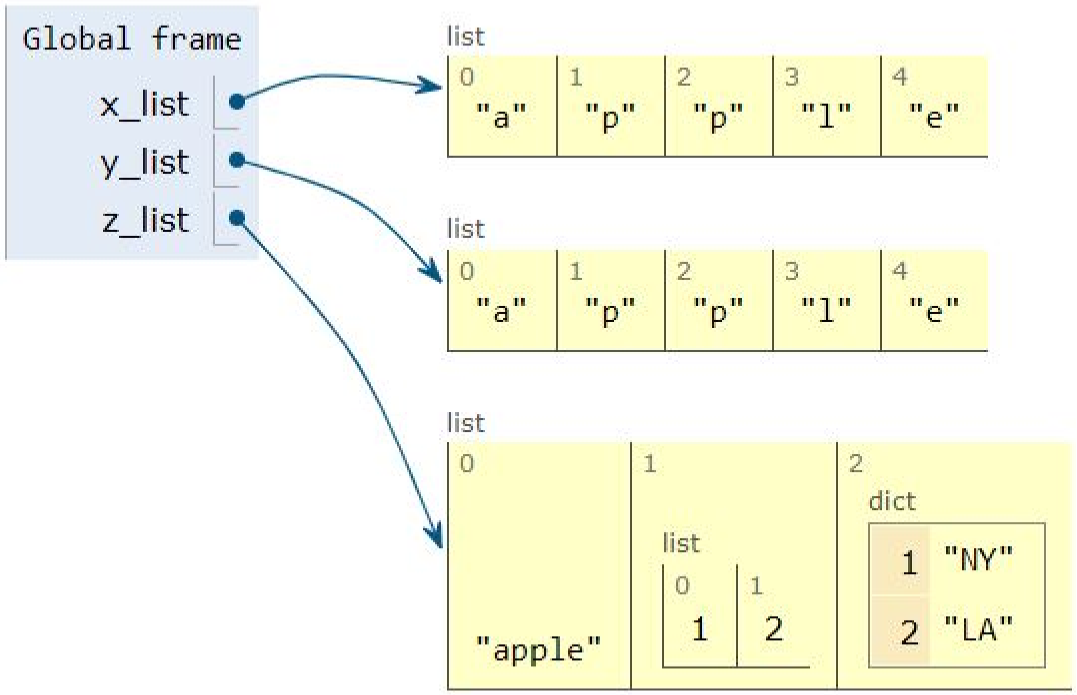
x_str "pineapple"

0	1	2	3	4	5	6	7	8
p	i	n	e	a	p	p	l	e

- object (not just an array)
- many built-in methods

A Python List

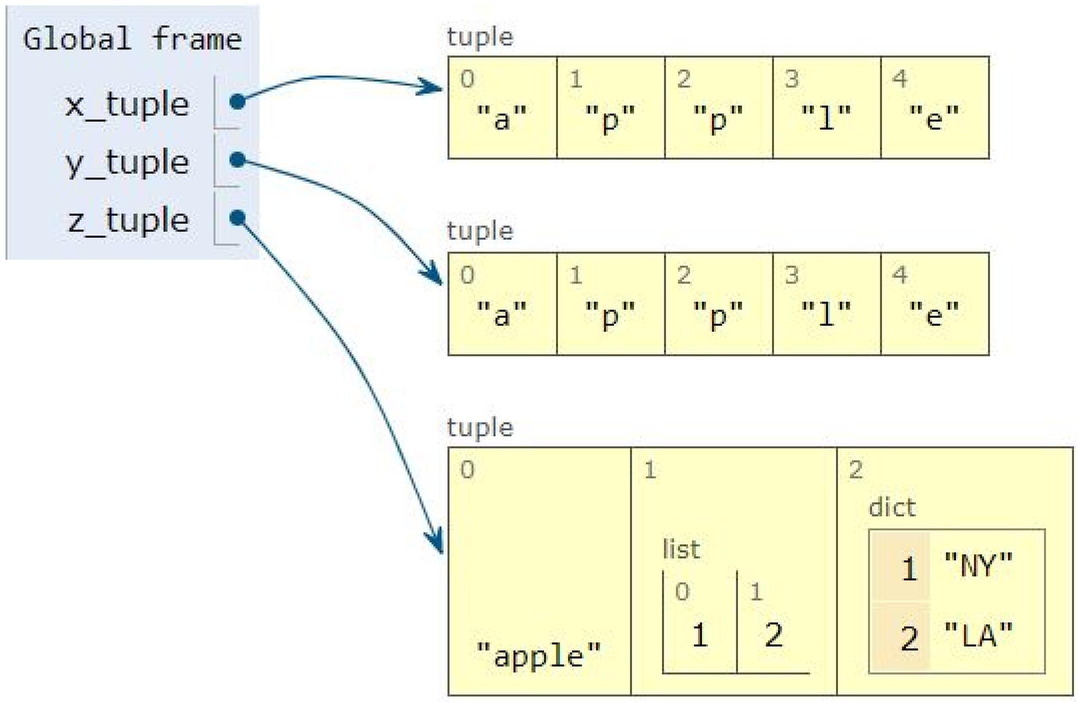
```
x_list = ['a','p','p','l','e']
y_list = list('apple')
z_list = ['apple', [1,2], {1:'NY', 2: 'LA'}]
```



- ordered collection
- can contain any objects

A Python Tuple

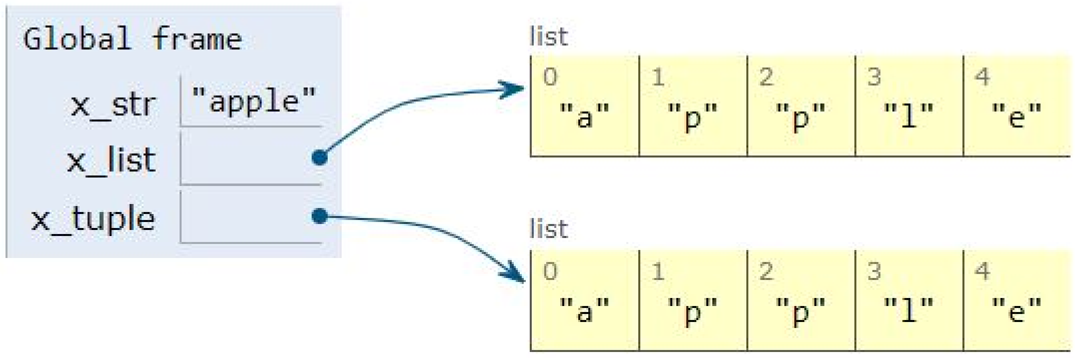
```
x_tuple = ('a','p','p','l','e')
y_tuple = tuple('apple')
z_tuple = ('apple', [1,2], {1:'NY', 2: 'LA'})
```



- ordered collection (like list)
- can contain any objects

Strings, Lists, Tuples

```
x_str   = 'apple'
x_list  = ['a','p','p','l','e']
x_tuple = ['a','p','p','l','e']
```



0	1	2	3	4
a	p	p	l	e

- ordered collections
- support indexing & slicing

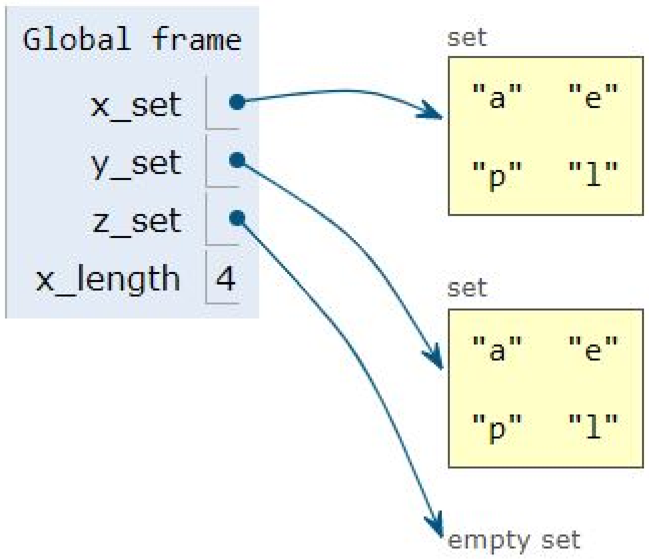
Exercise(s):

- show two different ways to construct *x_list* and *x_tuple* from *x_str*:

```
x_str = "apple"
```

A Python Set

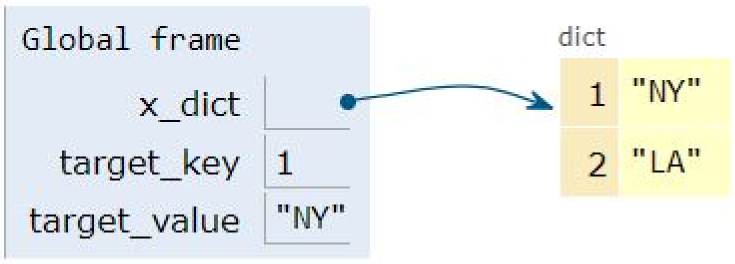
```
x_set = {'a','p','p','l','e'}
y_set = set('apple')
z_set = set()
```



- un-ordered, unique elements
- restrictions on elements

A Python Dictionary

```
x_dict = { 1: 'NY', 2: 'LA' }
target_key = 1
target_value = x_dict[target_key]
```

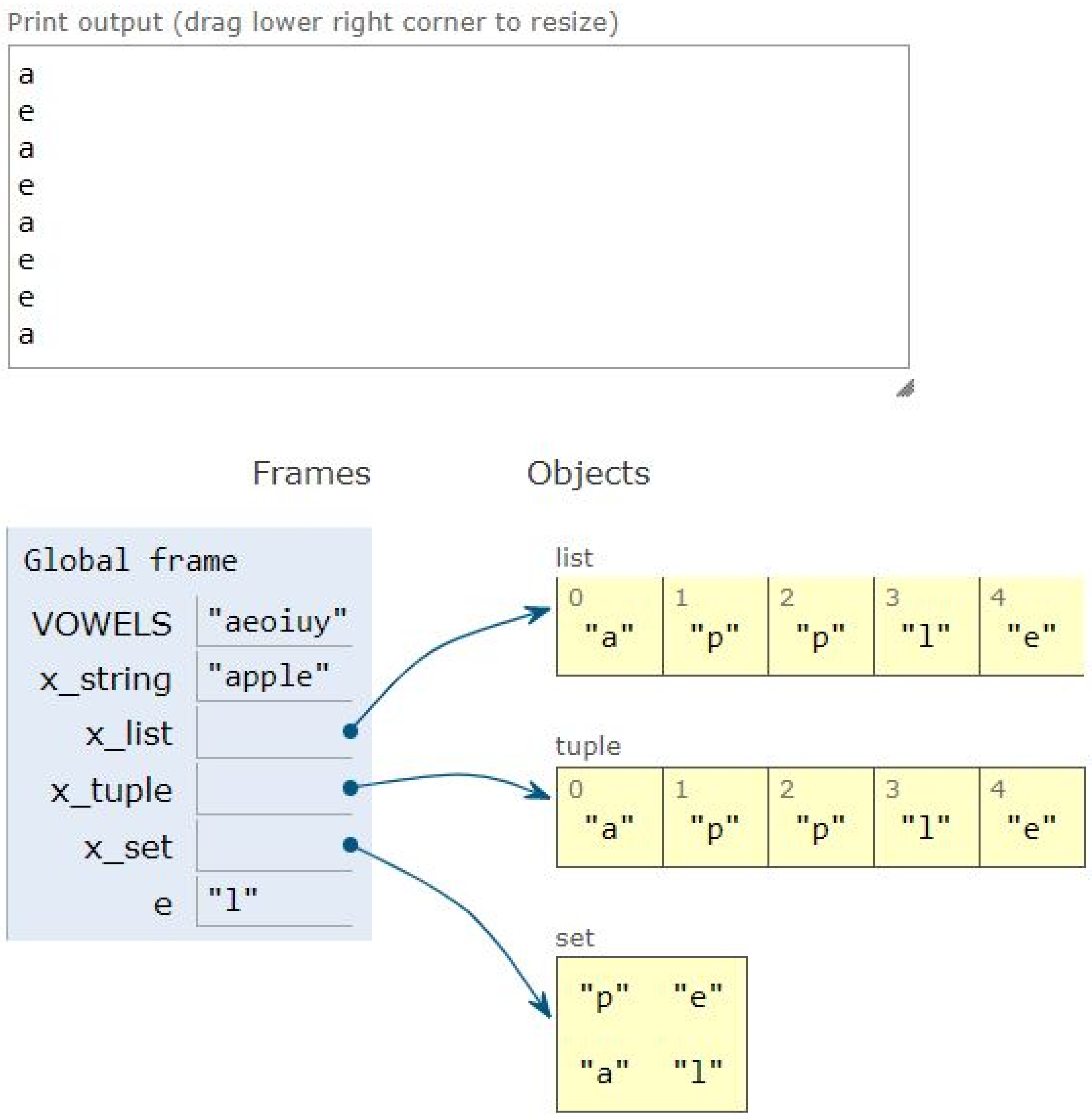


- collection of (*key, value*) pairs
- such pairs are called *items*
- built-in functions for *keys*, *values* and *items*

Iteration in Collections

```
VOWELS = 'aeoiuy'
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ('a','p','p','l','e')
x_set    = {'a','p','p','l','e'}

for e in x_string:
    if e in VOWELS:
        print(e, end = '')

for e in x_list:
    if e in VOWELS:
        print(e, end = '')

for e in x_tuple:
    if e in VOWELS:
        print(e, end = '')

for e in x_set:
    if e in VOWELS:
        print(e, end = '')
```

Iteration in Collections



Print output (drag lower right corner to resize)

```
a
e
a
e
a
e
e
a
```

Exercise(s):

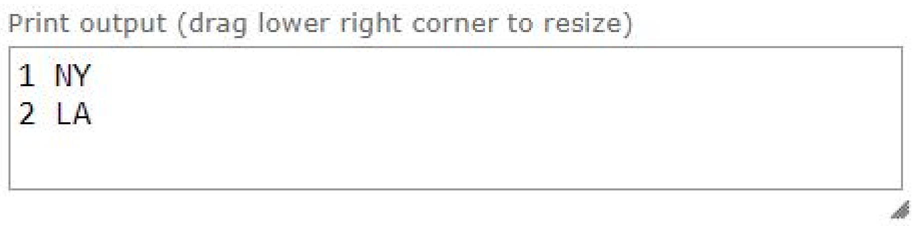
- write iterations to print consonants from the following collections:

```
x_str   = "automobile"
x_list  = list(x_str)
x_tuple = tuple(x_str)
x_set   = set(x_str)
```

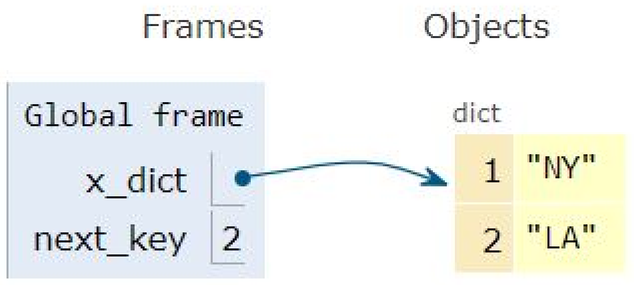
- why does *x_set* contain one less element than the other three collections?

Iteration in Dictionaries

```
# print dictionary keys and values
x_dict = {1: 'NT', 2: 'LA'}

for next_key in x_dict.keys():
    print(next_key, x_dict[next_key])
```

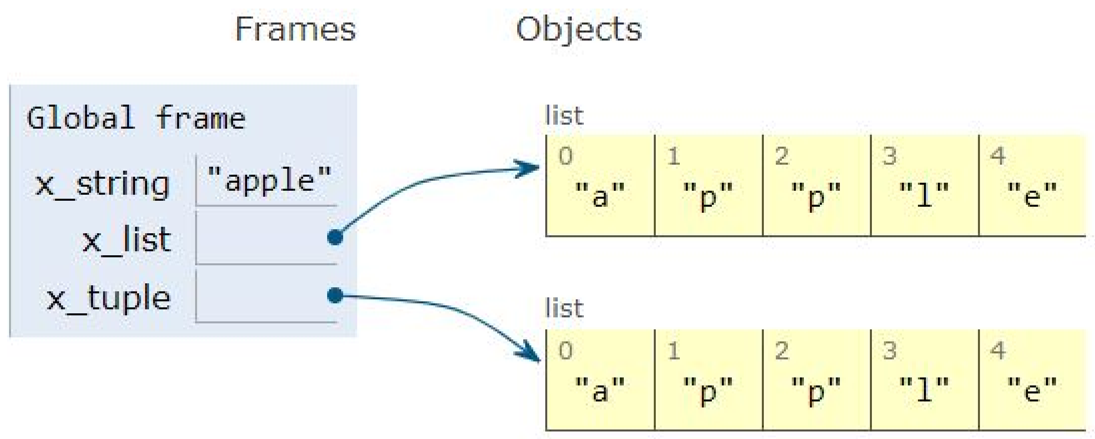

Print output (drag lower right corner to resize)

```
1 NY
2 LA
```



Ordered Collections

```
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ['a','p','p','l','e']
```



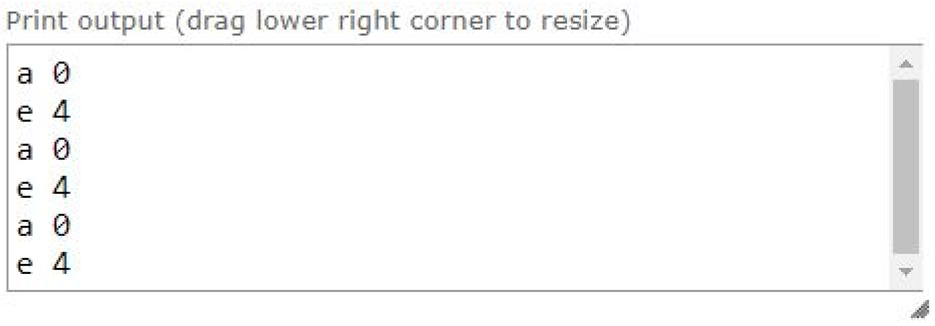
0	1	2	3	4
a	p	p	l	e

- enumerate
- indexing and slicing

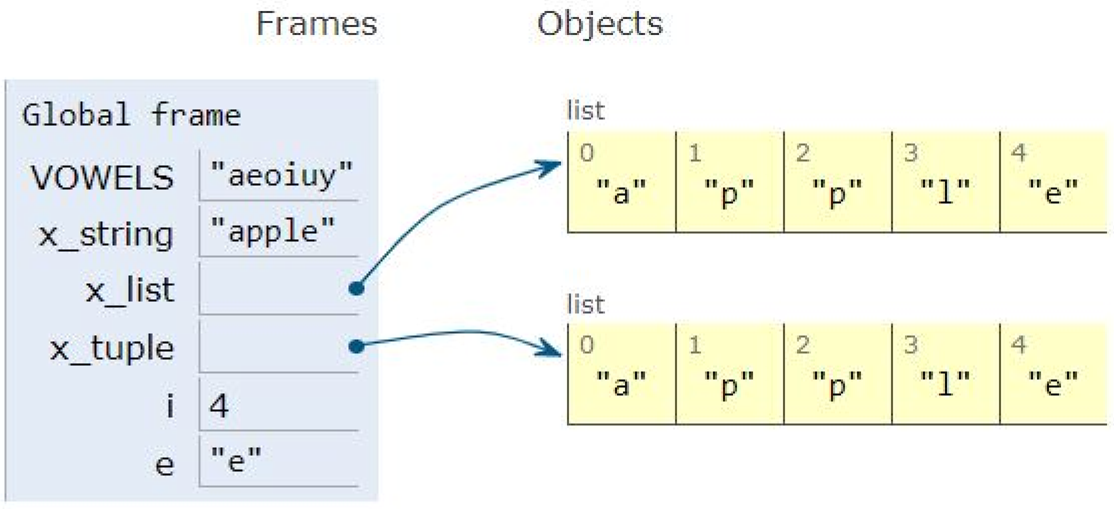
enumerate() in Collections

```
# print vowels and positions from collections
VOWELS   = 'aeoiuy'
x_string = 'apple'
x_list   = ['a','p','p','l','e']
x_tuple  = ['a','p','p','l','e']

for i,e in enumerate(x_string):
    e = x_string[i]
    if e in VOWELS:
        print(e,i)

for i,e in enumerate(x_list):
    e = x_list[i]
    if e in VOWELS:
        print(e,i)

for i,e in enumerate(x_tuple):
    e = x_tuple[i]
    if e in VOWELS:
        print(e,i)
```

enumerate() **in Collections** (cont'd)



```
a 0
e 4
a 0
e 4
a 0
e 4
```



- ordered collections only!

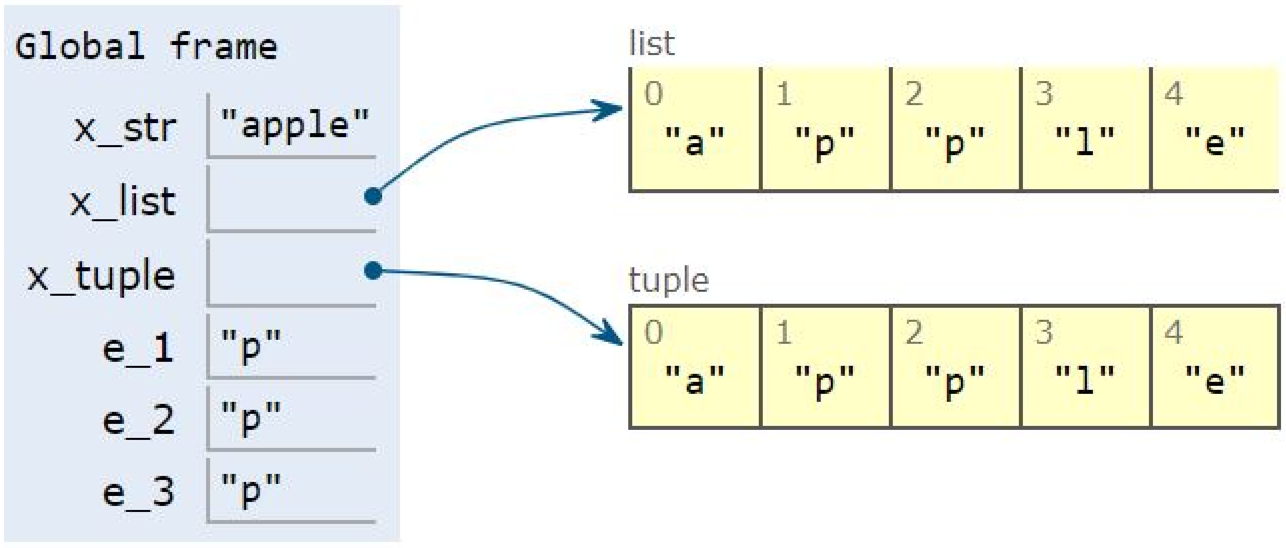
Exercise(s):

- use *enumerate*() iteration to print consonants and their positions from the following collections:

```
x_str   = "automobile"
x_list  = list(x_str)
x_tuple = tuple(x_str)
```

Indexing in Collections

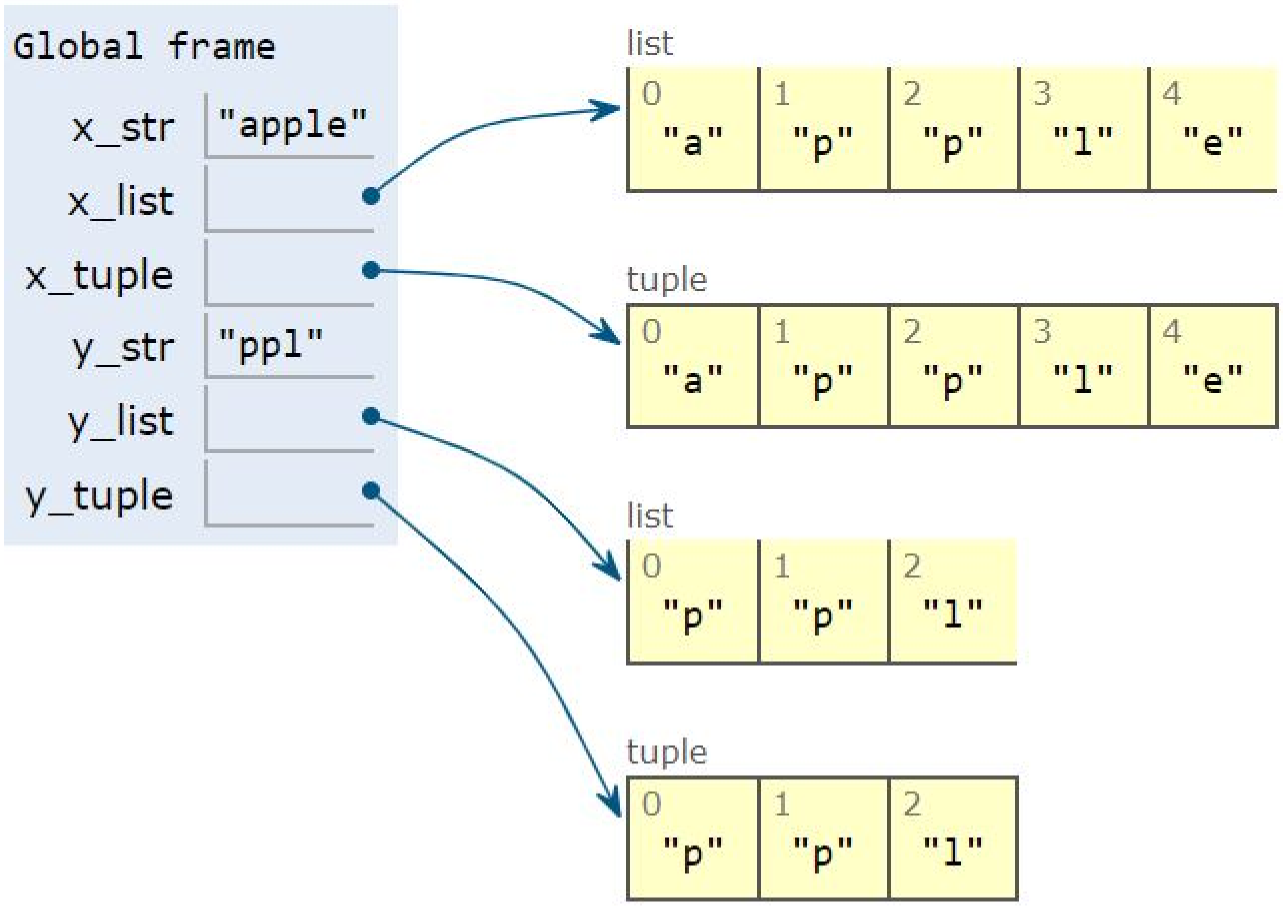
```
x_str   = 'apple'
x_list  = ['a','p','p','l','e']
x_tuple = ('a','p','p','l','e')
e_1     = x_str[1]
e_2     = x_list[1]
e_3     = x_tuple[1]
```



0	1	2	3	4
a	p	p	l	e

Slicing in Collections

```
x_str   = 'apple'
x_list  = list(x_str); y_tuple = tuple(x_str)
y_str   = x_str[1 : 4]
y_list  = x_list[1 : 4]
y_tuple = x_tuple[1 : 4]
```

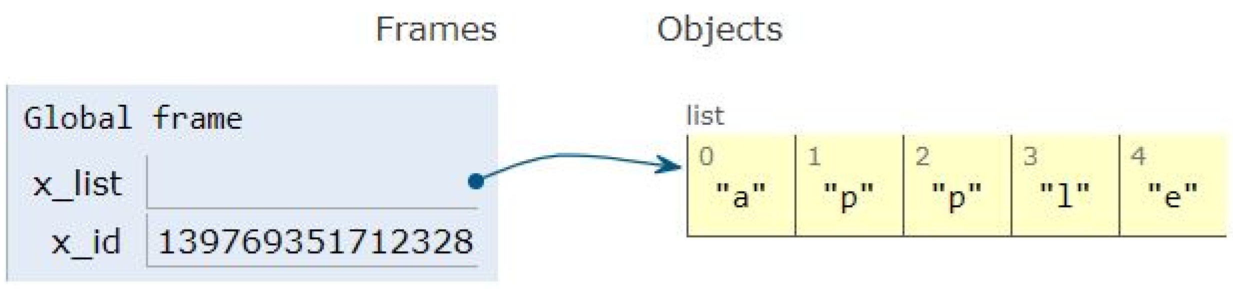


Mutability

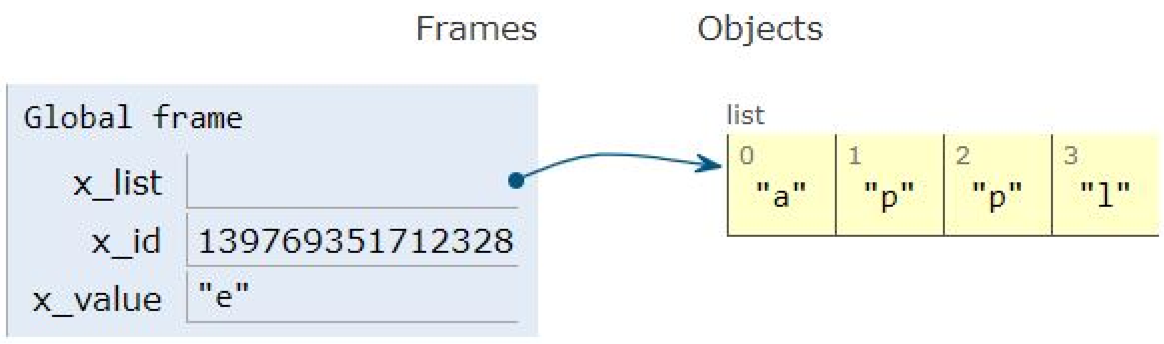
- mutable collections:
  1. list
  2. set
  3. dictionary
- immutable collections:
  1. strings
  2. tuples

List Mutability

```
x_list    = ['a', 'p', 'p', 'l', 'e']
x_id      = id(x_list)
```

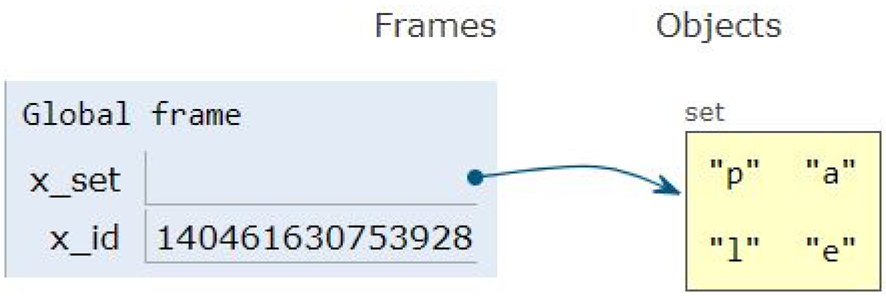


```
x_value   = x_list.pop(-1)   # remove last
x_id      = id(x_list)
```

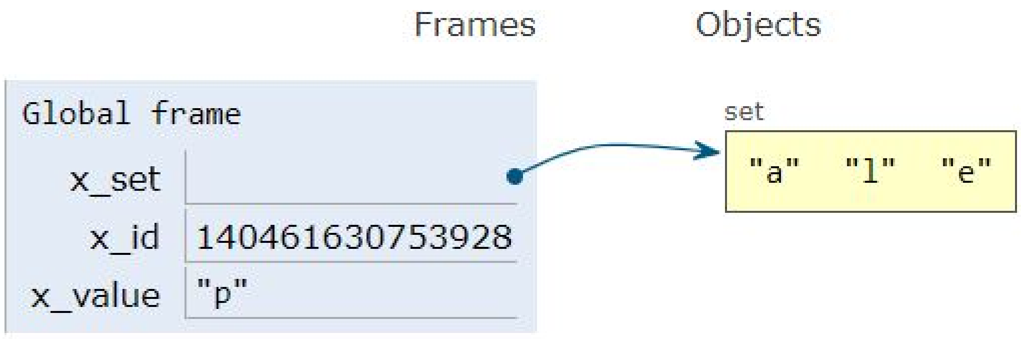


Set Mutability

```
x_set      = {'a', 'p', 'p', 'l', 'e'}
x_id       = id(x_set)
```

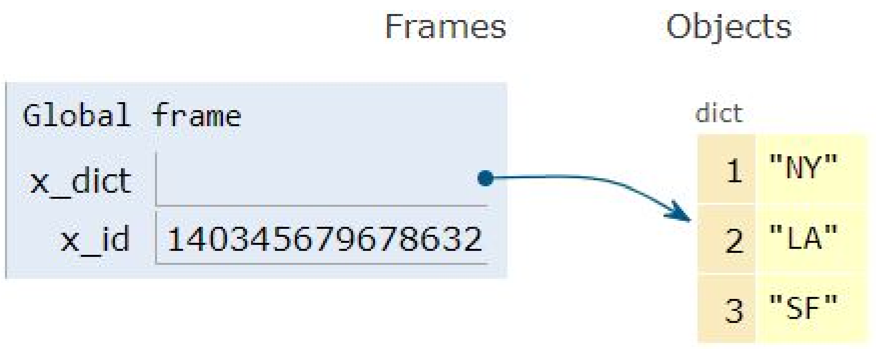


```
x_value    = x_set.pop()   # remove random
x_id       = id(x_set)
```

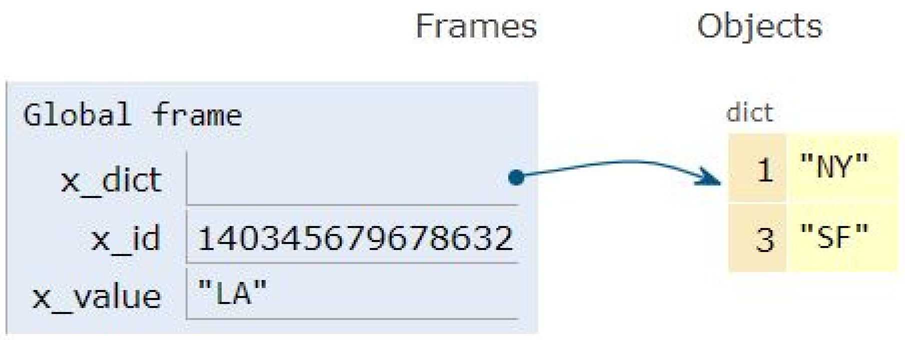


Dictionary Mutability

```
x_dict  = {1 : 'NY', 2: 'LA', 3: 'SF'}
x_id    = id(x_dict)
```



```
x_value = x_dict.pop(2)  # remove key = 2
x_id    = id(x_dict)
```



Exercise(s):

- is it possible to convert an immutable collection to a mutable one with same elements?
- is it possible to convert a mutable collection to an immutable one with same elements?

Summary of Collections

Collection	Ordered	Mutable
string	**yes**	**no**
list	**yes**	**yes**
tuple	**yes**	**no**
set	**no**	**yes**
dictionary	**no**	**yes**

- some variations:
  1. 'frozen' set (immutable)
  2. ordered dictionary

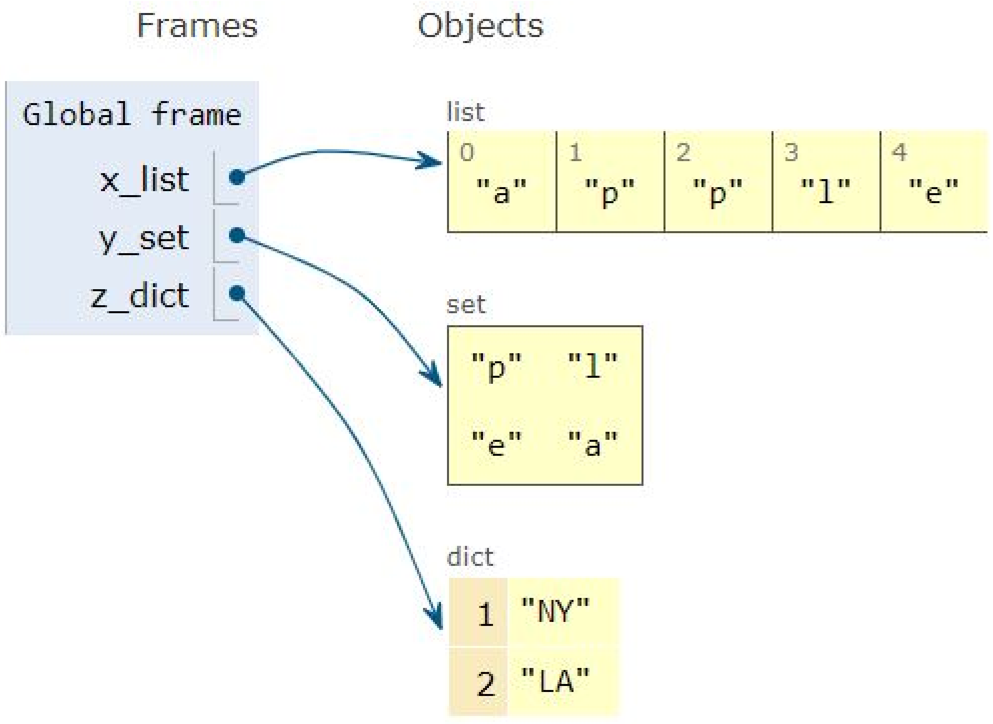
Common Methods

method	str	list	tuple	set	dict
clear	**n**	**y**	**n**	**y**	**y**
copy	**n**	**y**	**n**	**y**	**y**
count	**y**	**y**	**y**	**n**	**n**
index	**y**	**y**	**y**	**n**	**n**
pop	**n**	**y**	**n**	**y**	**y**
remove	**n**	**y**	**n**	**y**	**n**
update	**n**	**n**	**n**	**y**	**y**

- many *polymorphic* methods
- ordered: string, list, tuple
- mutable: list, set, dictionary

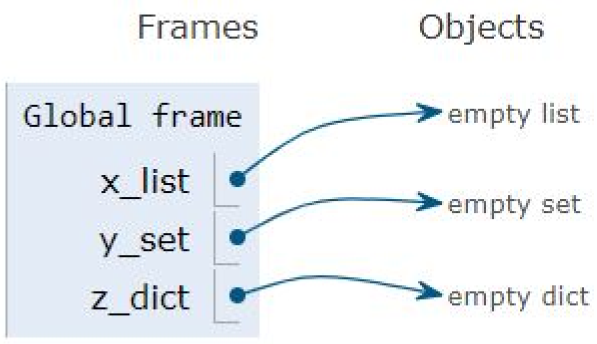
Collections: *clear*()

```
x_list = ['a', 'p', 'p', 'l', 'e']
y_set  = {'a', 'p', 'p', 'l', 'e'}
z_dict = {1 : 'NY', 2: 'LA'}
```



Collections: *clear*() (cont'd)

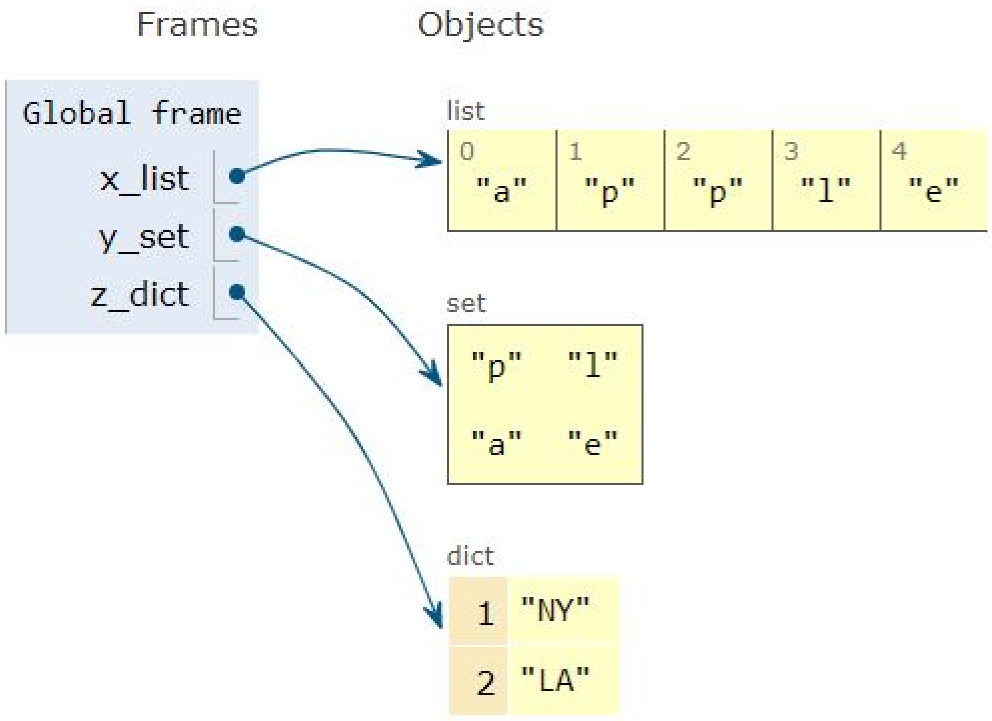
```
x_list.clear()
y_set.clear()
z_dict.clear()
```



- mutable collections
- method is done in-place

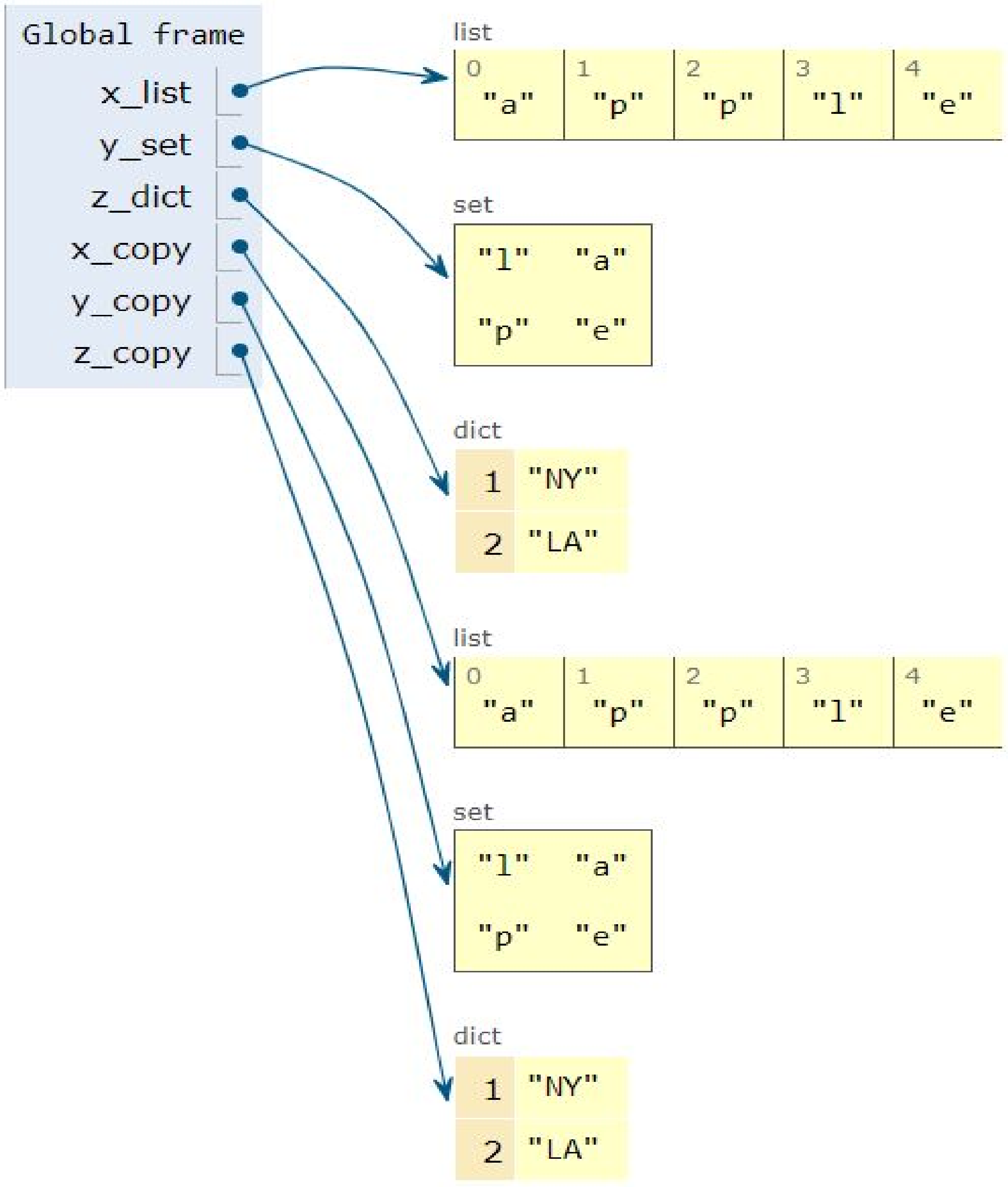
Collections: *copy*()

```
x_list = ['a', 'p', 'p', 'l', 'e']
y_set  = {'a', 'p', 'p', 'l', 'e'}
z_dict = {1 : 'NY', 2: 'LA'}
```



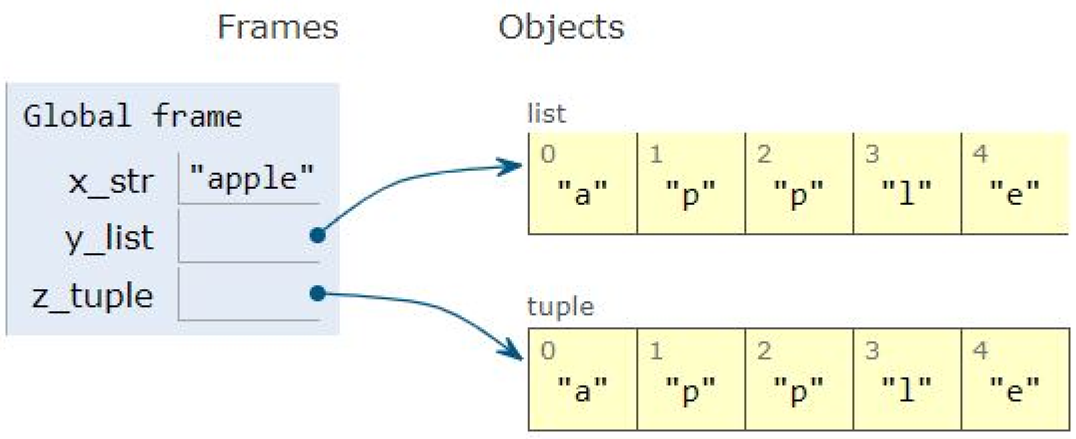
```
x_copy = x_list.copy()
y_copy = y_set.copy()
z_copy = z_dict.copy()
```

Collections: *copy*()



Collections: *count*()

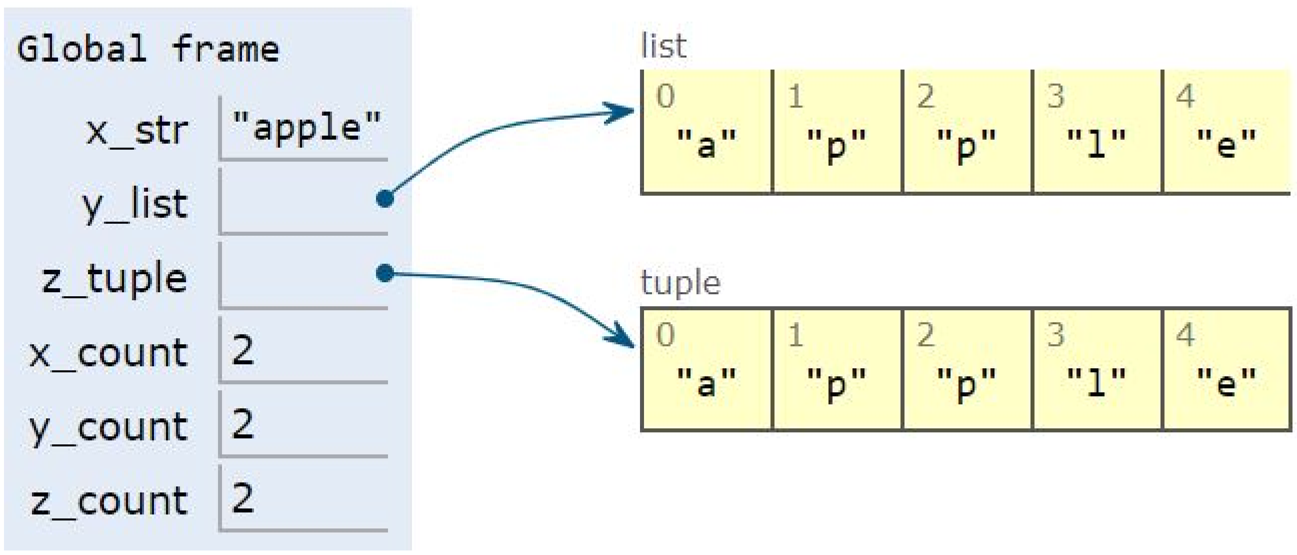
```
x_str   = 'apple'
y_list  = ['a', 'p', 'p', 'l', 'e']
z_tuple = ('a', 'p', 'p', 'l', 'e')
```



- count number of occurencies

Collections: *count*() (cont'd)

```
x_count = x_str.count('p')
y_count = y_list.count('p')
z_count = z_tuple.count('p')
```



Exercise(s):

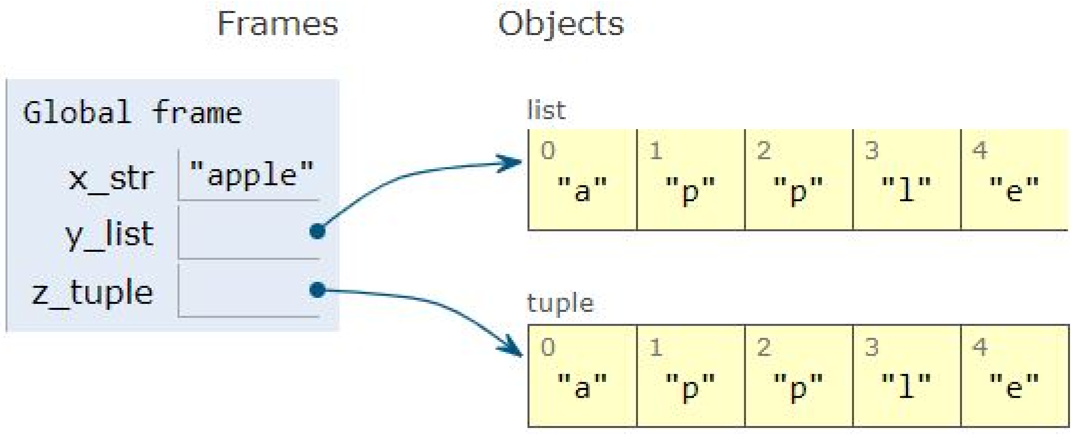
- count the number of occurencies of ”y” in

  ```
  x_str="monday tuesday"
  ```

- for each letter construct a dictionary of frequency counts

Collections: *index*()

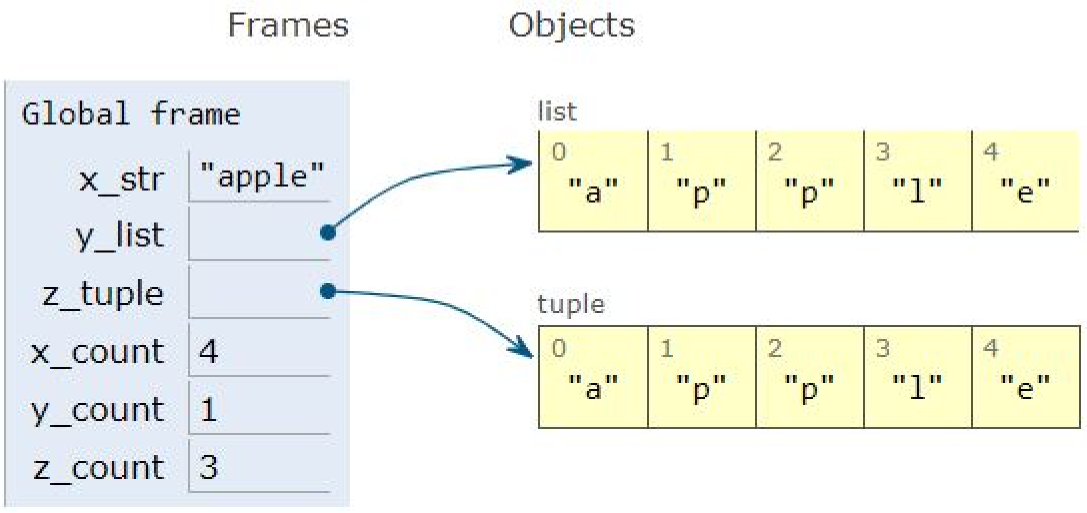
```
x_str   = 'apple'
y_list  = ['a', 'p', 'p', 'l', 'e']
z_tuple = ('a', 'p', 'p', 'l', 'e')
```



- ordered collections
- index of first ocuurency
- note: value must exist

Collections: *index*() (cont'd)

```
x_count = x_str.index('e')
y_count = y_list.index('p')
z_count = z_tuple.index('l')
```



Exercise(s):

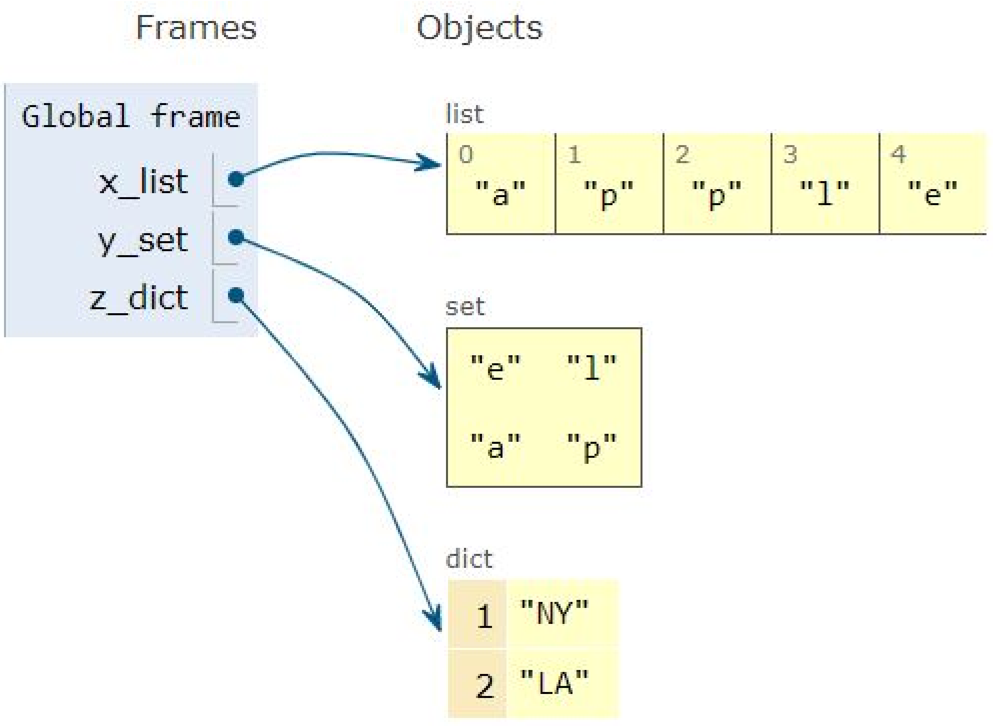
- compute the position of first ”a” in

  ```
  x_str="monday tuesday"
  ```

- compute the position of second ”a” in the same string

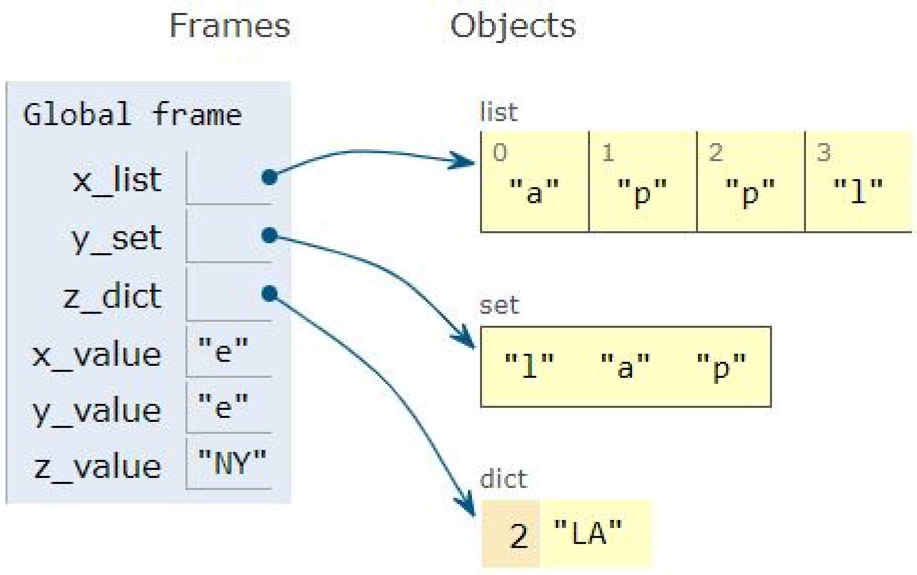
Collections: *pop*()

```
x_list = ['a', 'p', 'p', 'l', 'e']
y_set  = {'a', 'p', 'p', 'l', 'e'}
z_dict = {1 : 'NY', 2: 'LA'}
```



Collections: *pop*() (cont'd)

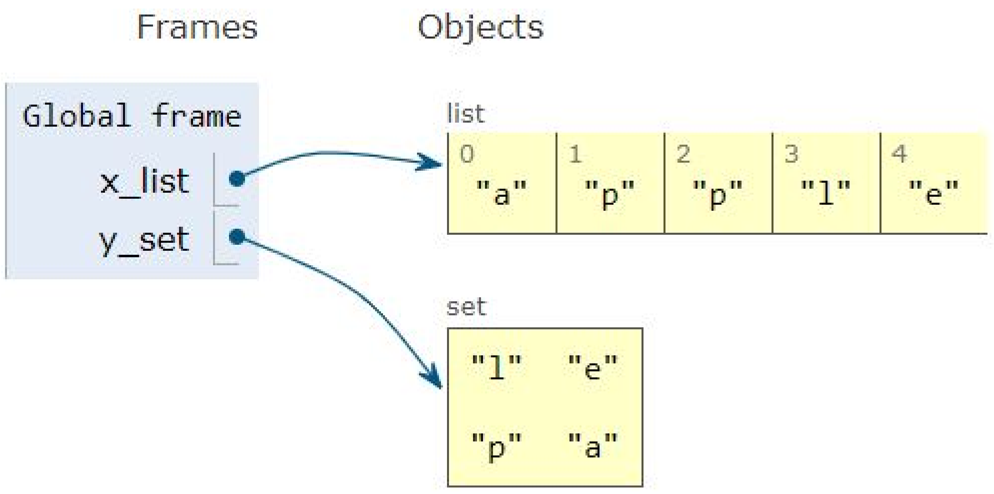
```
x_value = x_list.pop(-1) # last element
y_value = y_set.pop()     # random element
z_value = z_dict.pop(1)   # value for key=1
```



- mutable collections
- method is done in-place

Collections: *remove*()

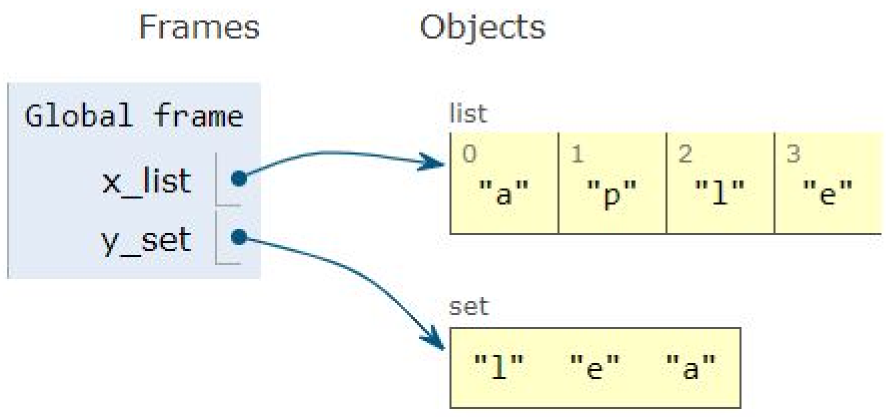
```
x_list = ['a', 'p', 'p', 'l', 'e']
y_set  = {'a', 'p', 'p', 'l', 'e'}
```



- remove by value

Collections: *remove*() (cont'd)

```
x_list.remove('p')
y_set.remove('p')
```



- mutable collections
- method is done in-place

Exercise(s):

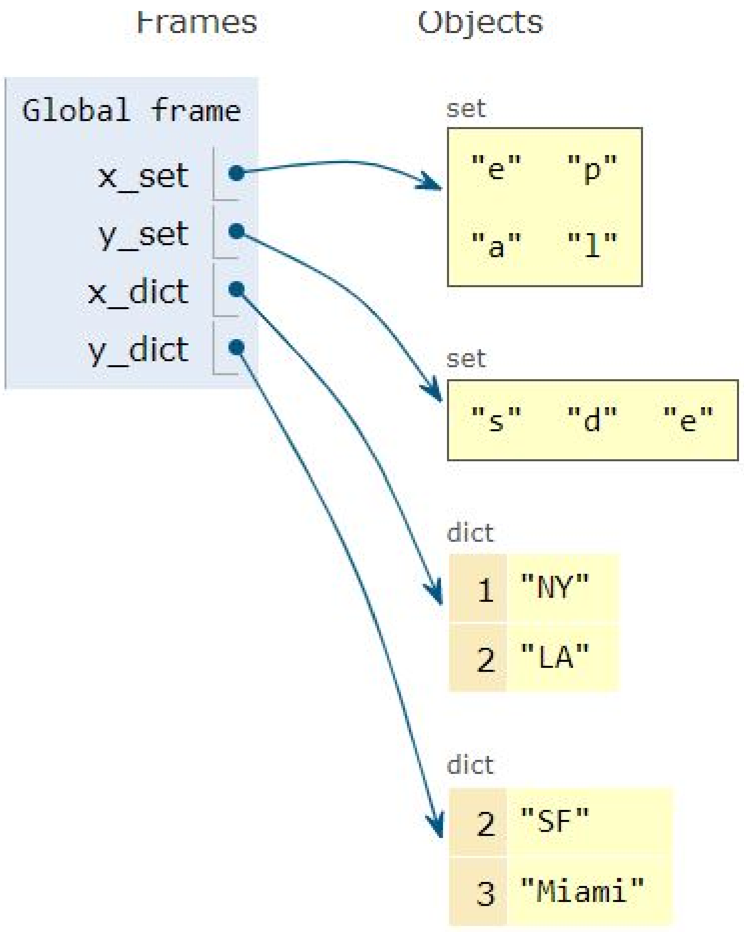
- remove even numbers from

  `x_list=[7,4,9,10,12,20,17]`
- remove even numbers from

  `x_set={7,4,9,10,12,20,17}`

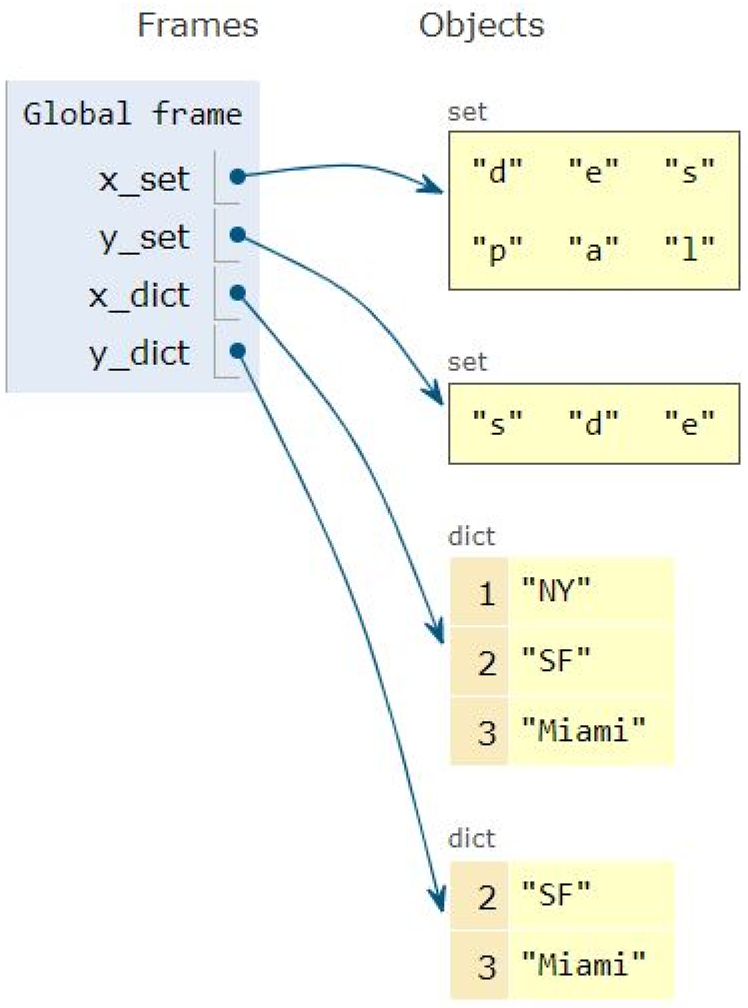
Collections: *update*()

```
x_set  = {'a', 'p', 'p', 'l', 'e'}
y_set  = {'s','e','e','d'}
x_dict = {1 : 'NY', 2: 'LA'}
y_dict = {2: 'SF', 3: 'Miami'}
```



Collections: *update*()

```
x_set.update(y_set)
x_dict.update(y_dict)
```



Exercise(s):

- use *update*() to transform *x_set* into *y_set*

```
x_set = {1, 2, 3, 4, 5}
y_set = {1, 2, 3, 7, 8}
```

Collections: Summary

- string, list, tuple, set, dict
- iterable objects
- support membership methods
- ordered: strings, lists, tuples
- indexing & slicing (ordered)
- mutable: lists, sets, dict
- many *polymorphic* methods