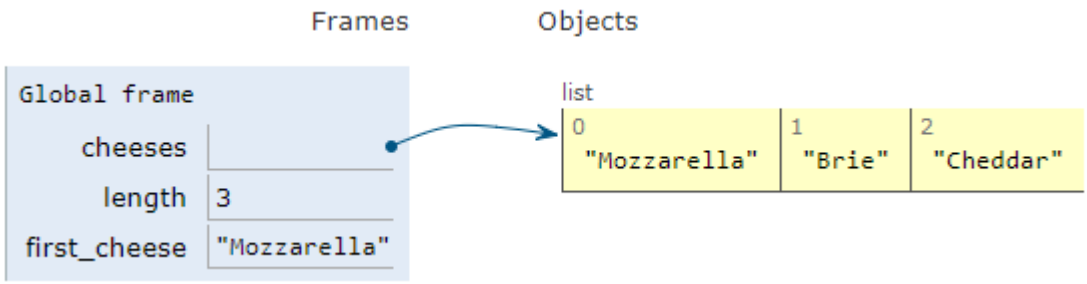
Collections

Python Collection Example

```
cheeses = ['Mozzarella', 'Brie', 'Cheddar']
length = len(cheeses)
first_cheese = cheeses[0]
for next_cheese in cheeses:
    print(next_cheese)
```



- elements: 'Mozzarella', 'Brie', 'Cheddar'
- length function: *len*()
- membership: *in*
- indexing/slicing: [0]
- iteration: *for*

Collection Mapping Types

- a data type that supports:
    1) membership (in/not in)
    2) size (len() function)
    3) iteration
- collections of (*key*, *value*) pairs
- built-in Python mapping types

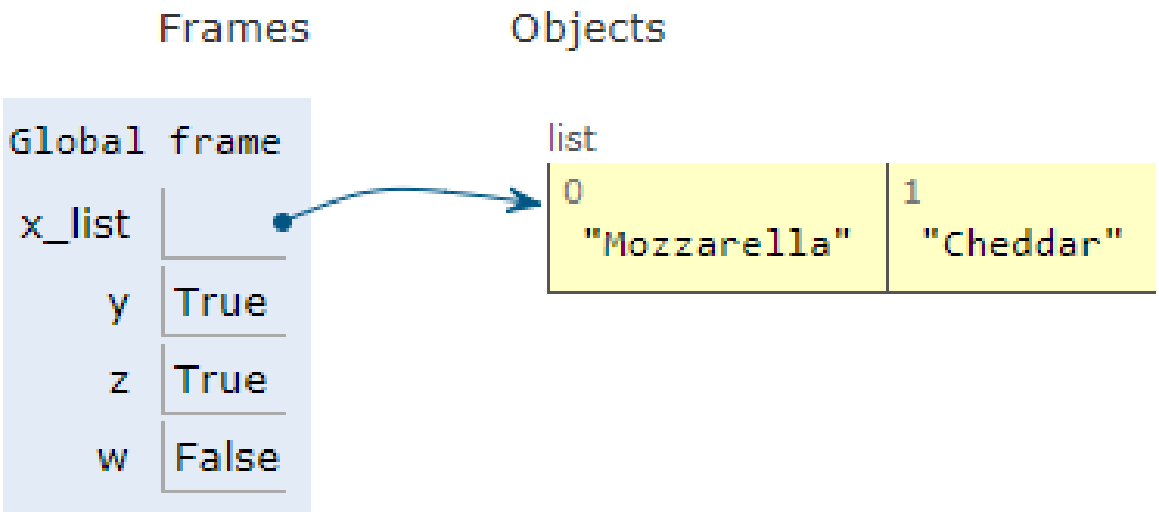
1. dictionary (unordered)
2. default dictionary (unordered)
3. ordered dictionary (ordered)

- keys contain “hashable” objects that must be unmutable (cannot contain lists)
- values can be mutable
- ordered dictionaries maintain order in which keys were put

The *in*/*not in* Operator

- check if object exists in a collection
- returns True or False

```
>>> x_list = ['Mozzarella', 'Cheddar']
>>> y = 'Cheddar' in x_list
>>> z = 'Brie' not in x_list
>>> w = 'Gouda' in x_list
```



collection **Data Types**

- sequence types

1) lists
2) tuples
3) named tuples

- set types

1) sets
2) frozen sets

- mapping types

1) dictionaries
2) default dictionaries
3) ordered dictionaries

Examples of Sequence Types

- built-in Python sequence types

1. bytes
2. bytearray
3. list
4. str
5. tuple
6. named tuple

```
from collections import namedtuple
x_bytearray = bytearray(buffer('Cheddar'))
x_list  = ['Mozzarella', 'Cheddar']
x_str = 'Mozzarella'
x_tuple = ('Mozzarella', 'Cheddar')
x_named_tuple  = namedtuple('Cheeses',
                            'Mozzarella', 'Cheddar')
```

Collection Sequence Types

- a data type that supports:

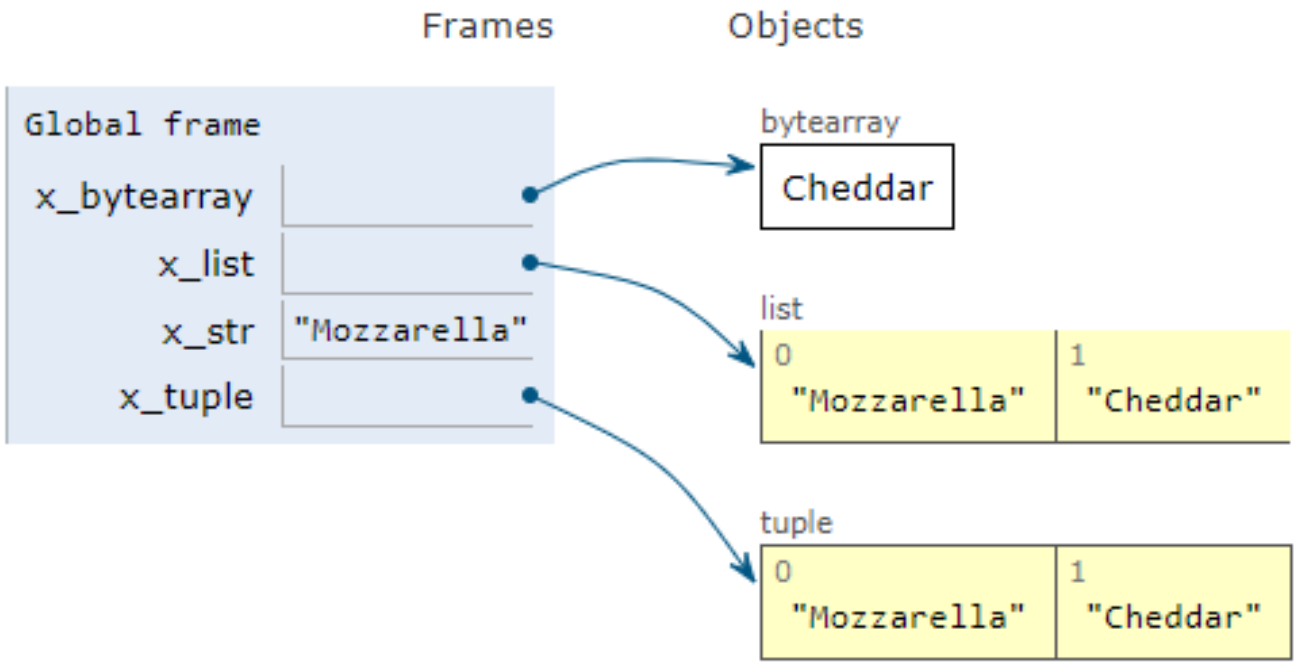
1) membership (in/not in)
2) size (len() function)
3) slicing ([])
4) iteration

- built-in Python sequence types

1. bytes
2. bytearray
3. list
4. str
5. tuple
6. named tuple

Bytearray, List, Str, Tuple

```
x_bytearray = bytearray(buffer('Cheddar'))
x_list  = ['Mozzarella', 'Cheddar']
x_str = 'Mozzarella'
x_tuple = ('Mozzarella', 'Cheddar')
```



Dictionaries

```
>>> from collections import defaultdict
>>> x_dict ={1:'Mozzarella', 2: 'Cheddar'}
>>> x_default = defaultdict(int, **x_dict)
>>> result = x_default_dict[5]
0
>>> result = x_dict[5]
KeyError: (5,)
```

- default dictionary allow non-existing key

Collection Set Types

- a data type that supports:
    1) membership (in/not in)
    2) size (len() function)
    3) iteration
    4) comparisons and bitwise

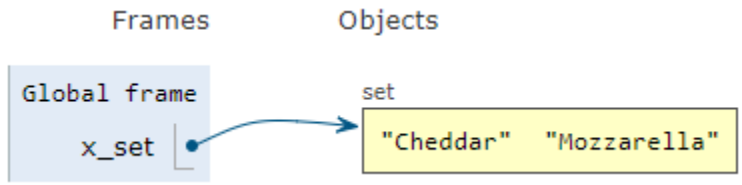
- built-in Python set types

1. sets
2. frozen sets

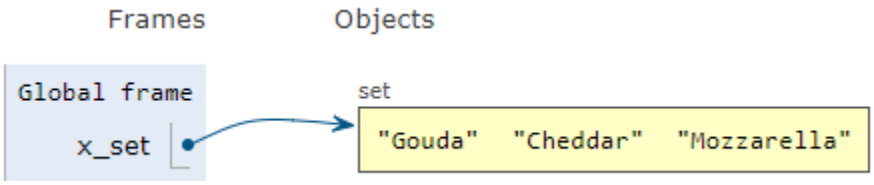
- contains “hashable” objects that must be unmutable (cannot contain lists)
- unordered and mutable (set)
- unordered and immutable (frozen set)

Sets in Detail

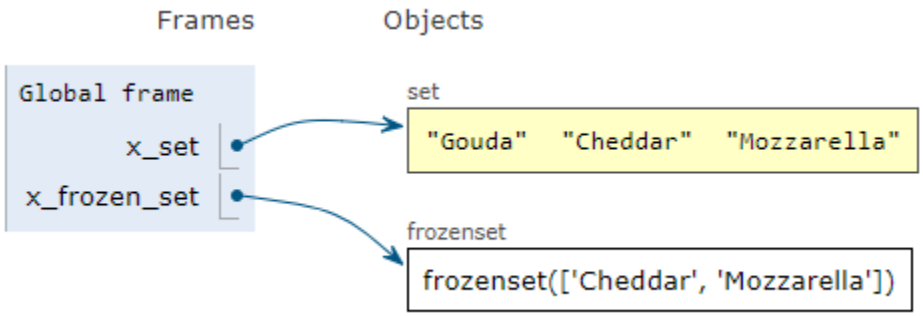
```
>>> x_set  = set(('Mozzarella', 'Cheddar'))
```




```
>>> x_set.add('Gouda')
```




```
>>> x_frozen_set = frozenset(('Mozzarella',
                              'Cheddar'))
```




```
>>> x_frozen_set.add('Gouda')
```

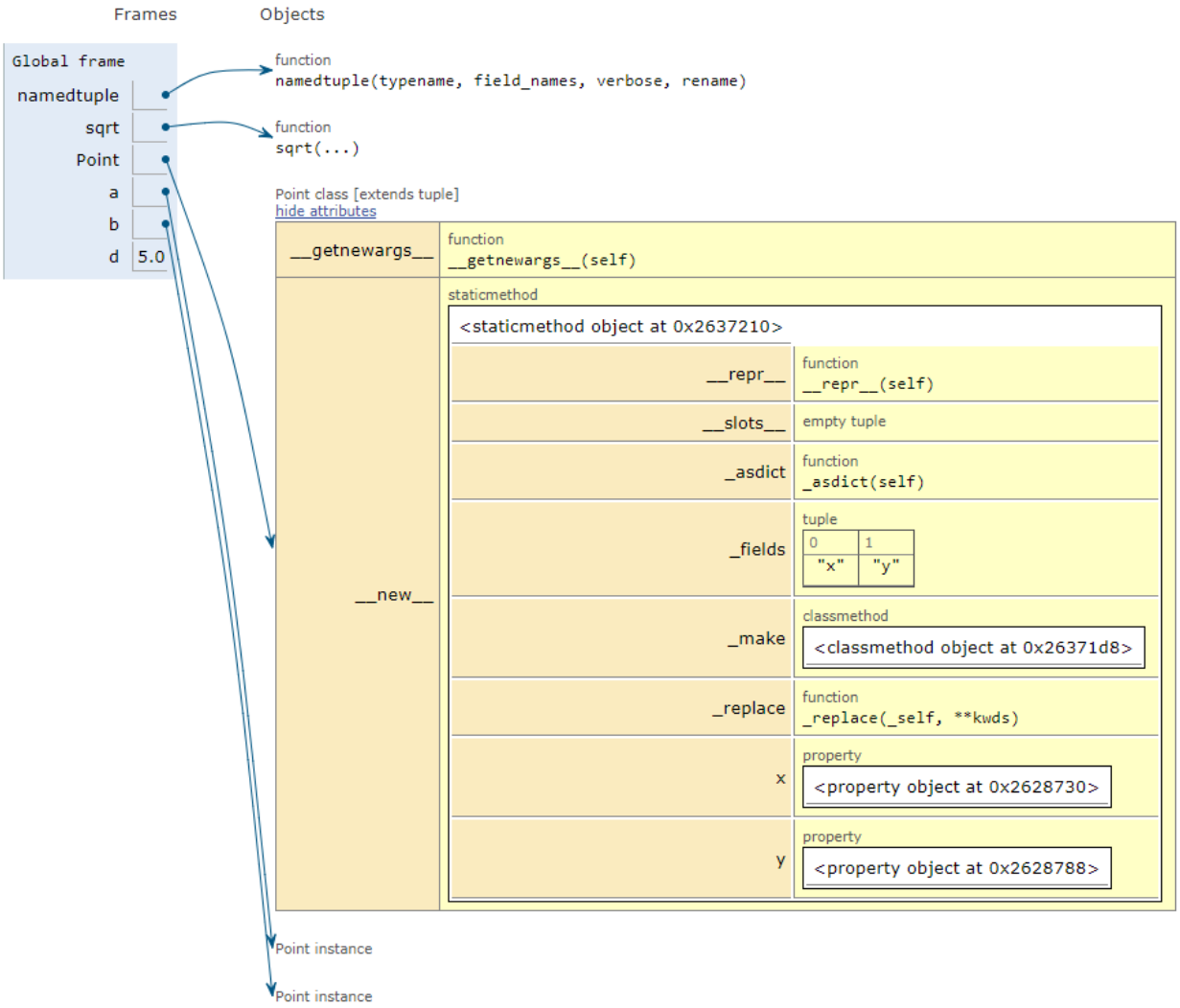
```
AttributeError: 'frozenset' object has no attribute
 'add'
```

Named Tuple

```
from collections import namedtuple
from math import sqrt
Point = namedtuple('Point', 'x y')
a = Point(2, 7)
b = Point(5, 11)
d = sqrt((a.x – b.x)**2 + (a.y – b.y)**2)
```

- can be referenced as standard tuples or objects (a[0] or a.x)
- immutable
- use for readability

Named Tuple in Detail



Iterators

- a data type that can return one item at a time is called *iterable*
- such methods have __iter__ method and __getitem__() method

Generators

- a function that produces a sequence

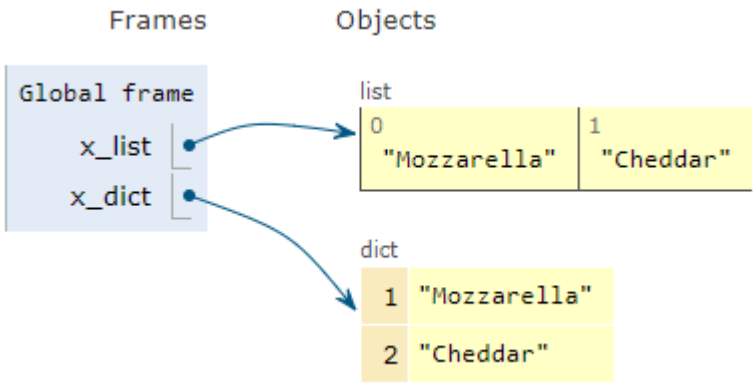
```
def compute_cubes(n):
    x = 0
    while x < n:
        yield x**3
        x = x + 1
```

```
>>> y = next_cubes(3)   # generator
>>> y.next()
0
>>> y.next()
1
>>> y.next()
8
>>> y.next()
None
```

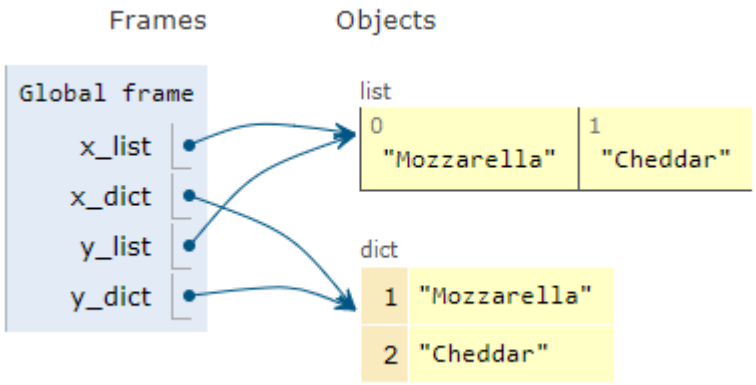
Copying Collections

- no copying takes place in assignment
- both variables point to the same object

```
>>> x_list =['Mozzarella', 'Cheddar']
>>> x_dict ={1:'Mozzarella', 2: 'Cheddar'}
```

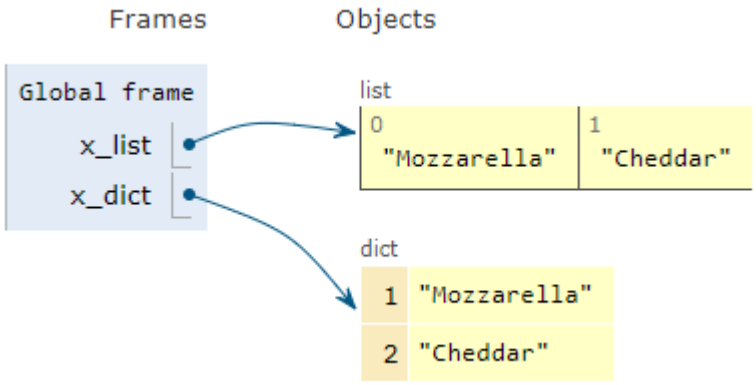



```
>>> y_list = x_list
>>> y_dict = x_dict
```

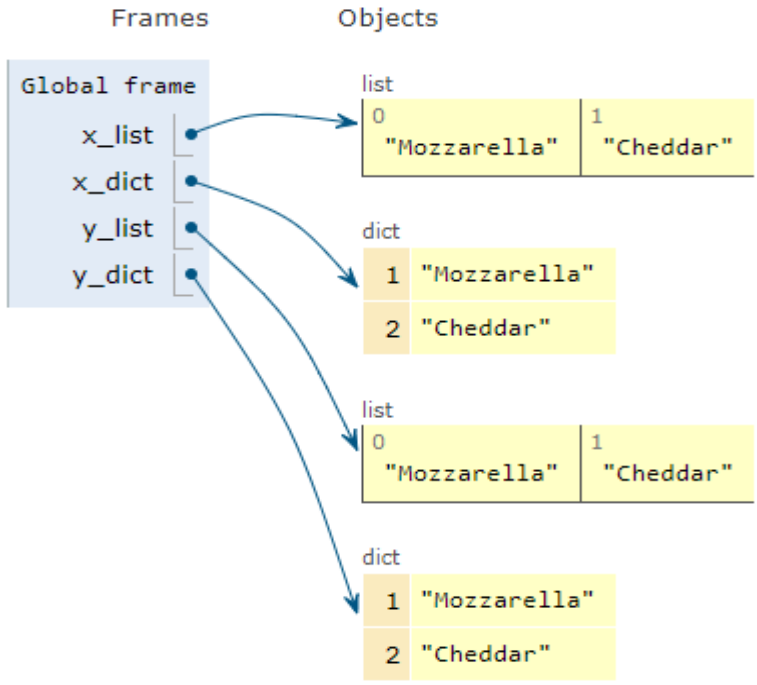


Copying Collections via Type

```
>>> x_list =['Mozzarella', 'Cheddar']
>>> x_dict ={1:'Mozzarella', 2: 'Cheddar'}
```

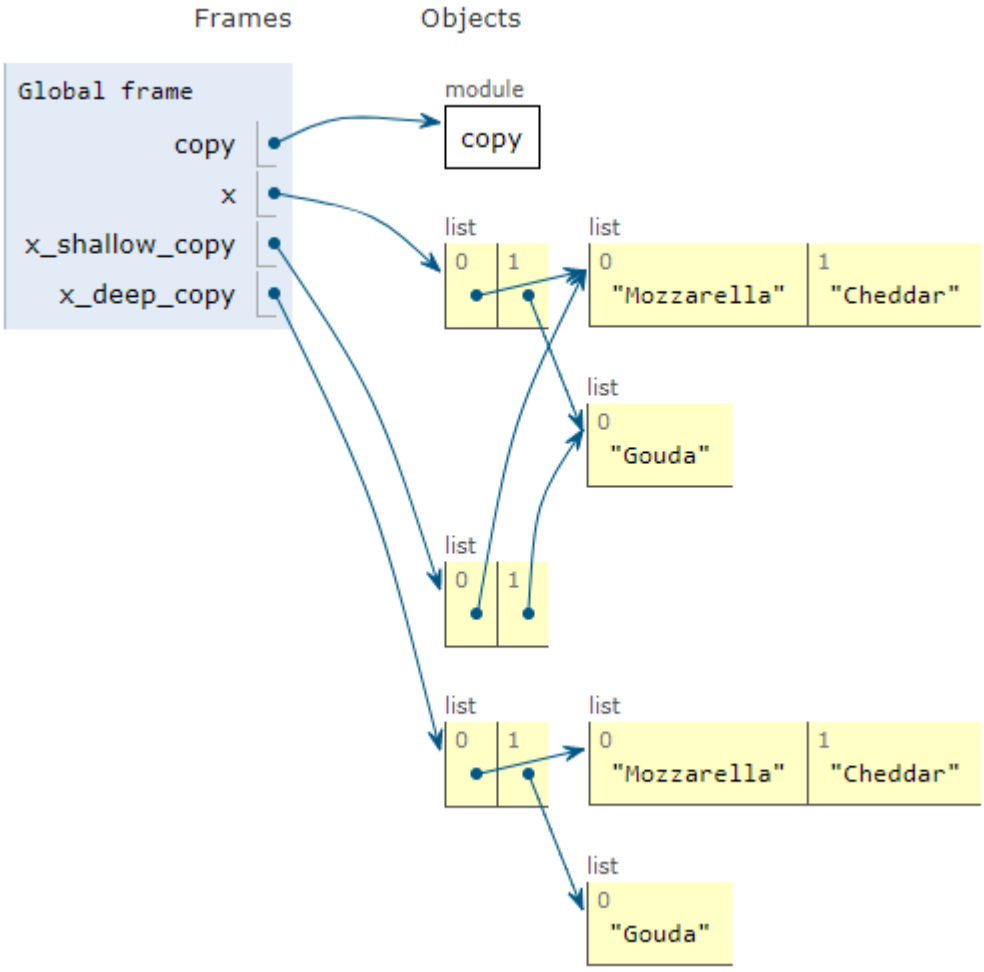



```
>>> y_list = list(x_list)
>>> y_dict = dict(x_dict)
```



Copy: Mutable Elements

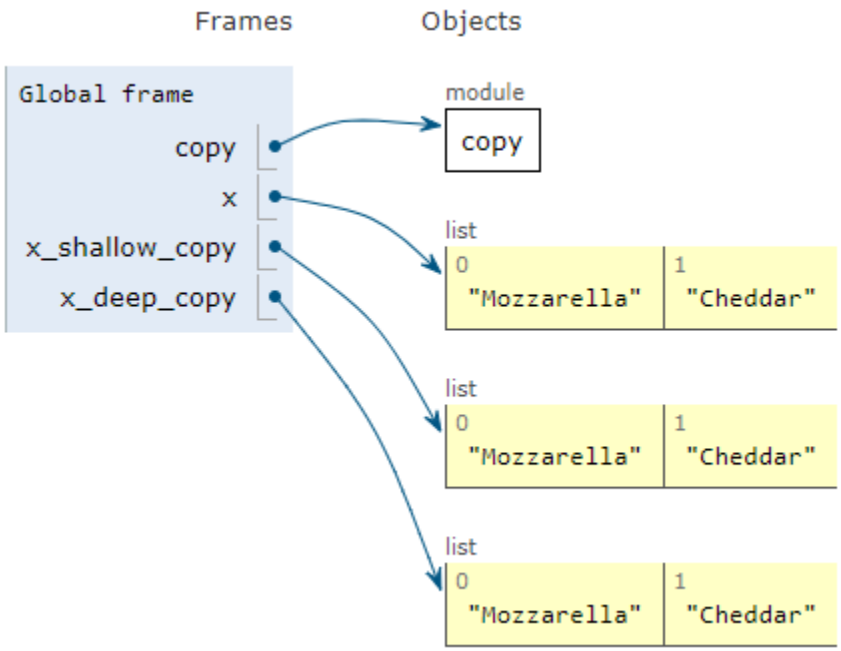
```
>>> import copy
>>> x = [['Mozzarella','Cheddar'], ['Gouda'] ]
>>> x_shallow_copy = x[:]
>>> x_deep_copy = copy.deepcopy(x)
```



- shallow copy is different from deep copy

Copy: Immutable Elements

```
>>> import copy
>>> x = ['Mozzarella', 'Cheddar']
>>> x_shallow_copy = x[:]
>>> x_deep_copy = copy.deepcopy(x)
```



- can use either shallow and deep copy

Review Problems

Interview Problem

- what is the difference between:
  (1) list
  (2) tuple
  (3) dictionary
  (4) set

Interview Problem

- what does *zip*() function do?

Interview Problem

- give an example to explain how collection membership is determined

Interview Problem

- what are generators?

Interview Problem

- explain the difference between *list* and *tuple* in terms of:
  (1) syntax
  (2) mutability
  (3) size
  (4) performance
  (5) usage

Interview Problem

- explain the difference between *list* and *dictionary* in terms of:
  (1) syntax
  (2) referencing (indexing)
  (3) ordering
  (4) hashing

Interview Problem

- what is the use of *enumerate*() in Python?

Interview Problem

- give some examples of sequences in Python

Interview Problem

- what are different methods to copy an object?

Interview Problem

- define and explain the difference between:
  (1) shallow copy
  (2) deep copy