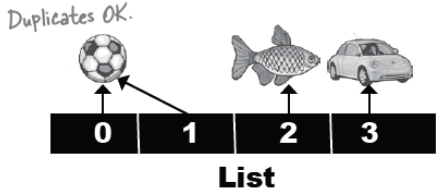
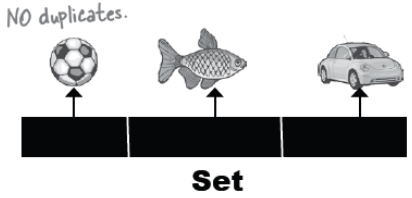
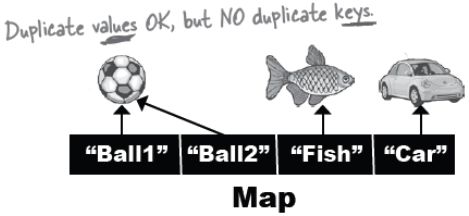
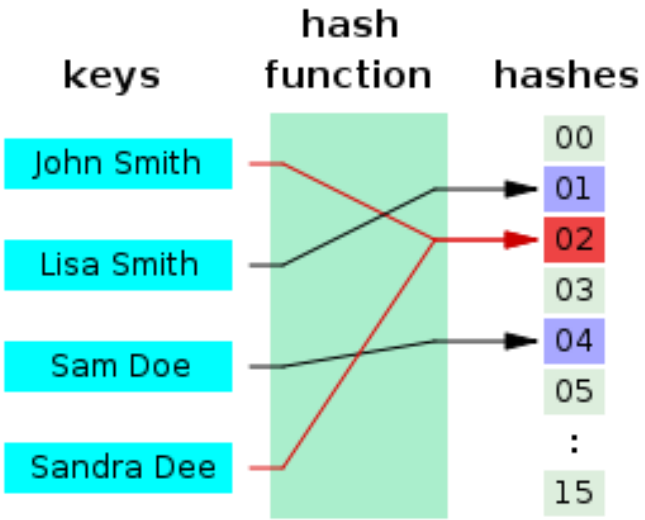
Dictionaries (and Sets)









Overview

- Dictionaries
  - Creating
  - Accessing
  - Common Operations
- Sets
  - Creating
  - Common Operations

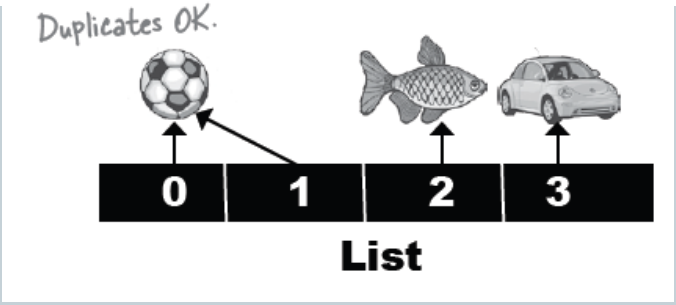
Mapping keys to values

- Common problem: map one thing to another
  - Often from a very large table of key/value pairs
  - Often we want to do this really fast

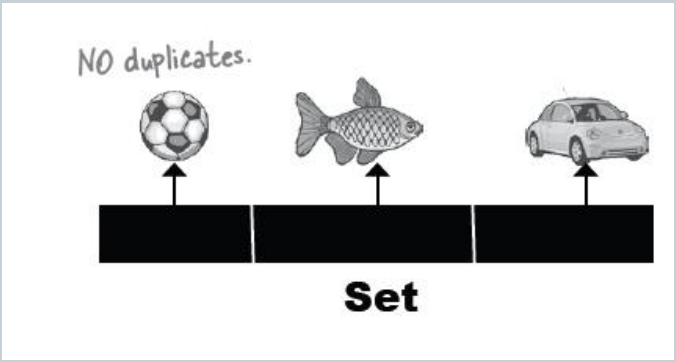
Application	Purpose	Key	Value
phone book	look up phone number	name	phone number
dictionary	look up word	word	definition
zip code	map city to a zip code	city	zip code
login screen	check user knows password	username	password
file system	find file on disk	filename	location on disk
web search	find all relevant pages	search keywords	list of pages
DNS	find IP address given URL	URL	IP address
reverse DNS	find URL given an IP address	IP address	URL

Collections: useful data types for storing stuff

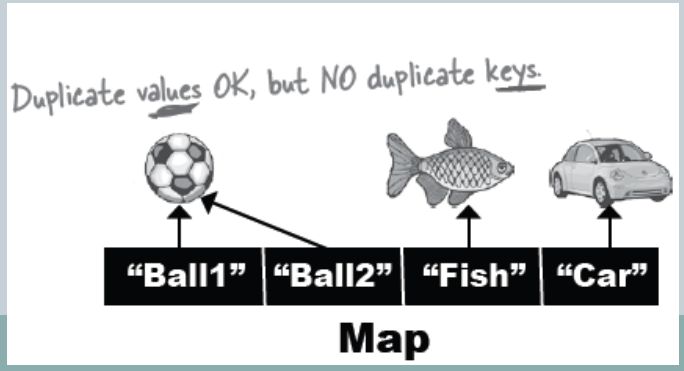
- List: Knows about the order of things



- Set: Knows if something is in the set or not



- Map: Knows how to get from a key to a value



Dictionary Example 1

- Goal: Map a domain name to an IP address

```
domainNameServer = {
    “katie.mtech.edu" : “10.33.73.166",
    “codemt.org" : “10.33.73.165",
    "www.google.com" : “216.58.193.68",
    "google.com" : " 216.58.193.68 ",
    }

domainNameServer[" katie.mtech.edu "] = "127.0.0.1"


str = input("Enter the domain name: ")
print(str + " -> " + domainNameServer[str])
```

Create a dictionary of domain names and IP addresses

Replace existing value of **10.33.73.166** with 127.0.0.1

Returns the value for a key, error if key is not found.

keys and values

- Key must be immutable
  - strings, integers, tuples are fine
  - lists are NOT immutable
- Value can be anything

Dictionary Example 2

- Goal: Type animal, play sound

```
import winsound

andThe__Says = {
    "cow" : "cow.wav",
    "frog" : "frog.wav",
    "CSCI Students" : "yay.wav"
    }

critter = input("What animal do you want to hear? ")

if critter == "cow":
    winsound.PlaySound(andThe__Says["cow"], winsound.SND_FILENAME)
elif critter == "frog":
    winsound.PlaySound(andThe__Says["frog"], winsound.SND_FILENAME)
elif critter == "CSCI Students":
    winsound.PlaySound(andThe__Says["CSCI Students"], winsound.SND_FILENAME)
else:
    winsound.PlaySound("explosion.wav", winsound.SND_FILENAME)
```

collections but not a sequence

- dictionaries are collections but they are not sequences such as lists, strings or tuples
  - there is no order to the elements of a dictionary
  - in fact, the order (for example, when printed) might change as elements are added or deleted.
- So how to access dictionary elements?

Access dictionary elements

Access requires `[ ]`, but the *key* is the index!

```
my_dict={}
```

- an empty dictionary

```
my_dict['bill']=25
```

- added the pair 'bill':25

```
print(my_dict['bill'])
```

- prints `25`

Dictionaries are mutable

- Like lists, dictionaries are a mutable data structure
  - you can change the object via various operations, such as index assignment

```
my_dict = {'bill':3, 'rich':10}
print(my_dict['bill'])    # prints 3
my_dict['bill'] = 100
print(my_dict['bill'])    # prints 100
```

again, common operators

Like others, dictionaries respond to these

- `len(my_dict)`
  - number of key:value **pairs** in the dictionary
- `element in my_dict`
  - boolean, is `element` a **key** in the dictionary
- `for key in my_dict:`
  - iterates through the **keys** of a dictionary

fewer methods

Only 9 methods in total. Here are some

- `key in my_dict`
  does the key exist in the dictionary
- `my_dict.clear()` – empty the dictionary
- `my_dict.update(yourDict)` – for each key in `yourDict`, updates `my_dict` with that key/value pair
- `my_dict.copy` - copy
- `my_dict.pop(key)` – remove key, return value

Dictionary content methods

- `my_dict.items()` – all the key/value pairs
- `my_dict.keys()` – all the keys
- `my_dict.values()` – all the values

They return what is called a *dictionary view*.

- the order of the views correspond
- are dynamically updated with changes
- are iterable

Views are iterable

```
for key in my_dict:
    print(key)
```

- prints all the keys

```
for key,value in my_dict.items():
    print (key,value)
```

- prints all the key/value pairs

```
for value in my_dict.values():
    print (value)
```

- prints all the values

Dictionary iteration example program

```
candy = {
    "Snickers" : 3,
    "Mars" : 1,
    "Butterfinger" : 3,
    "Reese's Pieces" : 10
    }

print("All keys:")
for key in candy:
    print(key)


print("\nAll values:")
for value in candy.values():
    print(value)


print("\nAll (key, value) pairs:")
for key, value in candy.items():
    print(key, value)
```

```
All keys:
Snickers
Mars
Butterfinger
Reese's Pieces
```

```
All values:
3
1
3
10
```

```
All (key, value) pairs:
Snickers -> 3
Mars -> 1
Butterfinger -> 3
Reese's Pieces -> 10
```

Order will be in the order of entry.

Frequency of words in list
3 ways

membership test

```
count_dict = {}
for word in word_list:
    if word in count_dict:
        count_dict[word] += 1
    else:
        count_dict[word] = 1
```

Example: Histogram.py

```
seq = input("Enter a word: ")

d = dict()
for element in seq:
    if element not in d:
        d[element] = 1
    else:
        d[element] += 1
print (d)
```

Sets

Sets, as in Mathematical Sets

- in mathematics, a set is a collection of objects, potentially of many different types
- in a set, no two elements are identical. That is, a set consists of elements each of which is unique compared to the other elements
- there is no order to the elements of a set
- a set with no elements is the empty set

Creating a set

Set can be created in one of two ways:

- constructor: `set(iterable)` where the argument is iterable

```
my_set = set('abc')
my_set → {'a', 'b', 'c'}
```

- shortcut: `{}`, braces where the elements have no colons (to distinguish them from dicts)

```
my_set = {'a', 'b','c'}
```

Diverse elements

- A set can consist of a mixture of different types of elements

```
my_set = {'a',1,3.14159,True}
```

- as long as the single argument can be iterated through, you can make a set of it

no duplicates

- duplicates are automatically removed

```
my_set = set("aabbccdd")
print(my_set)
```

→ `{'a', 'c', 'b', 'd'}`

example

```
# Creates an empty set
null_set = set()
print(null_set)

# No colons means set, not dictionary
a_set = {1, 2, 3, 4}
print(a_set)

# Duplicates are ignored
b_set = {1, 1, 2, 2, 2}
print(b_set)

# Different data types are ok
c_set = {'a', 1, 2.5, (5,6)}
print(c_set)

# Order is not maintained
a_set = set("abcd")
print(a_set)
```

common operators

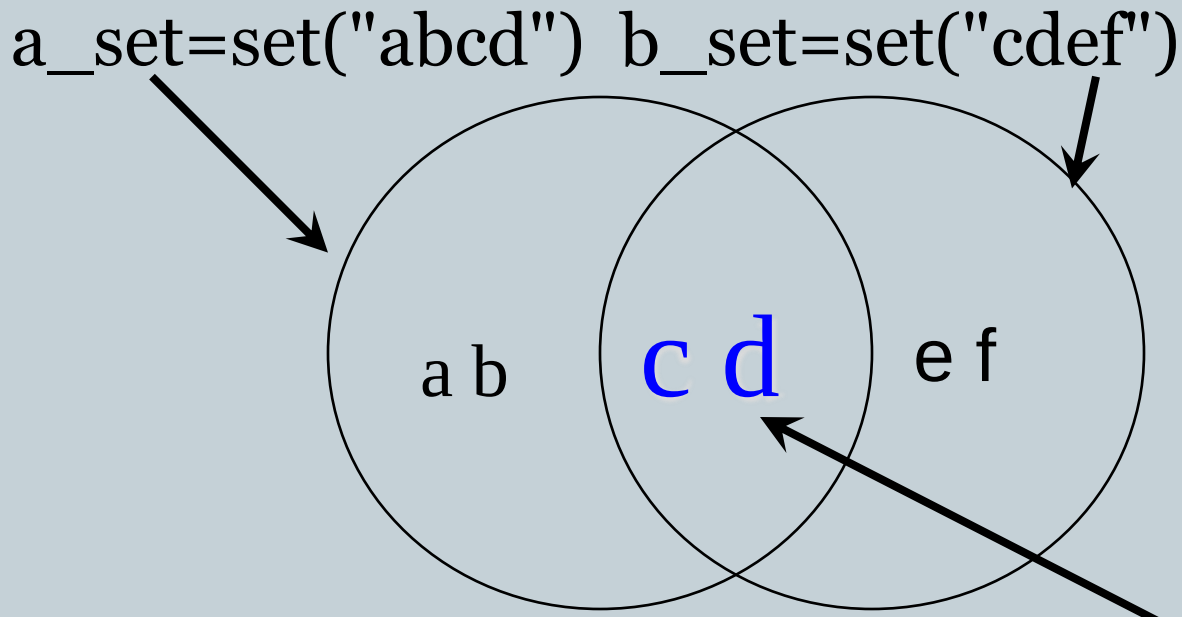
Most data structures respond to these:

- `len(my_set)`
  - the number of elements in a set
- `element in my_set`
  - boolean indicating whether element is in the set
- `for element in my_set:`
  - iterate through the elements in `my_set`

Set operators

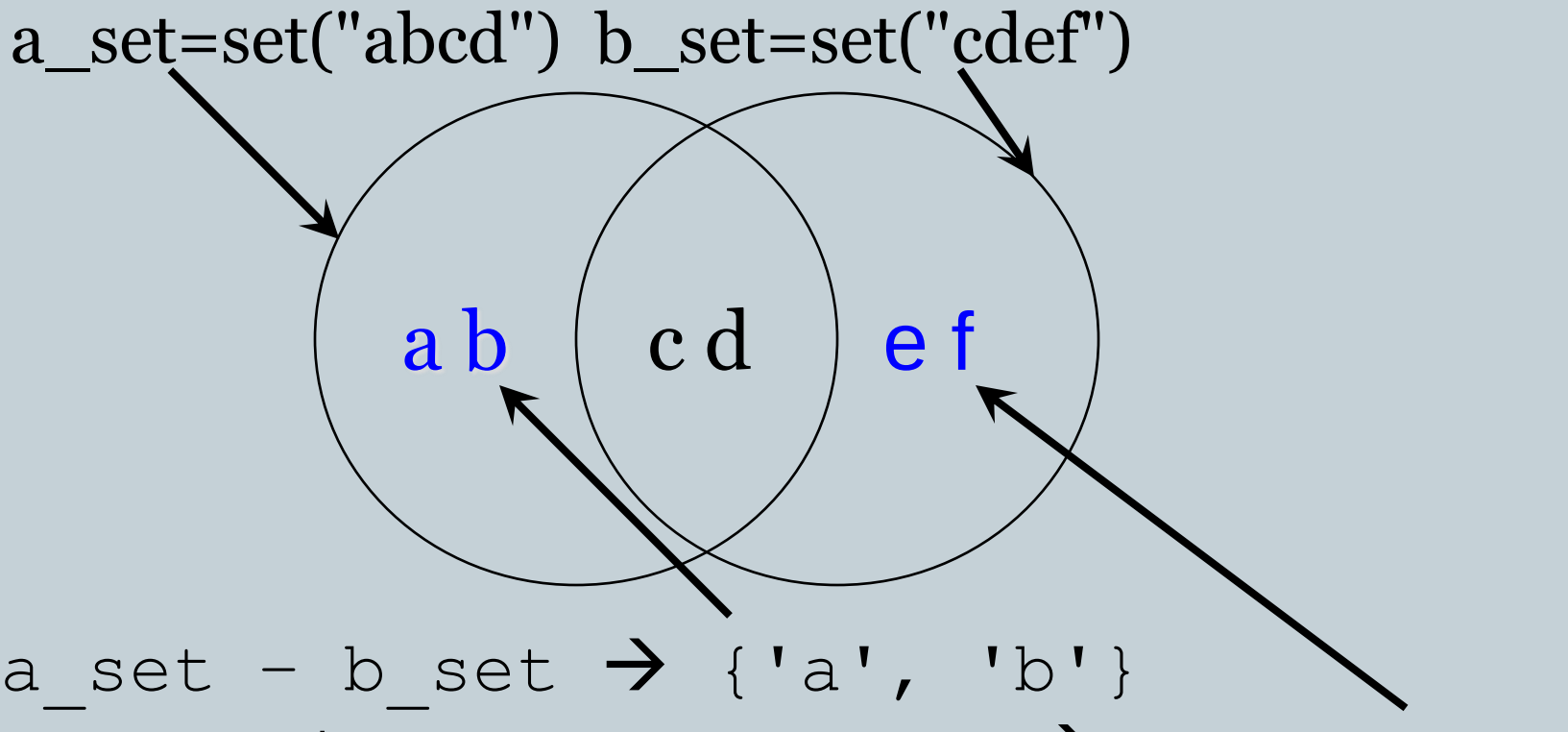
- The set data structure provides some special operators that correspond to the operators you learned in middle school.
- These are various combinations of set contents
- These operations have both a method name and a shortcut binary operator

method: intersection, op: &



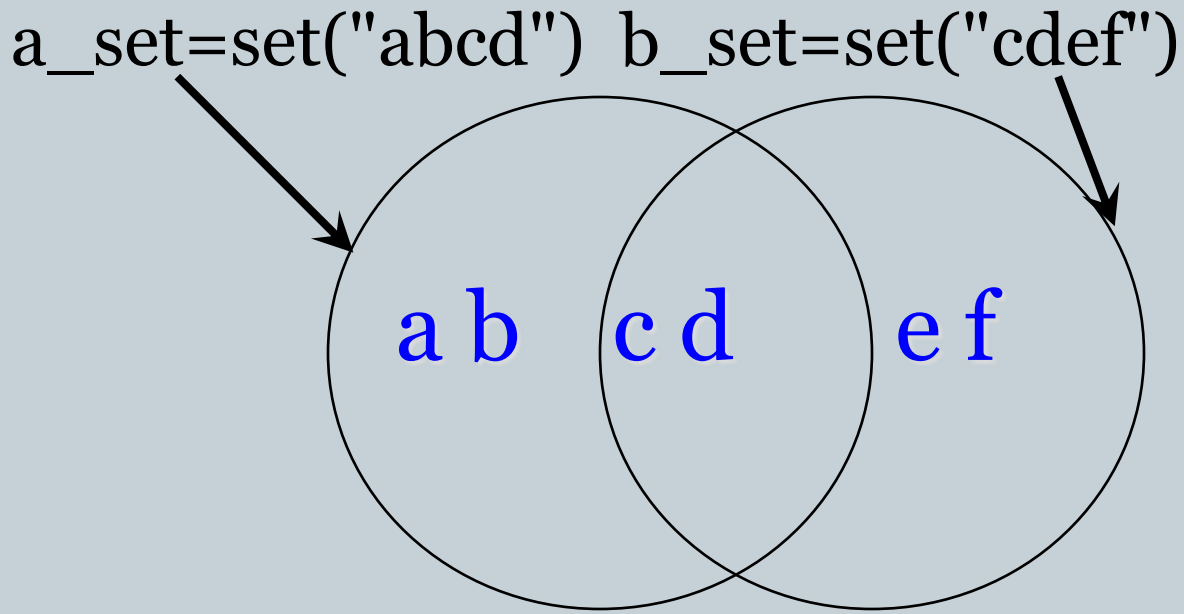

```
a_set & b_set → {'c', 'd'}
b_set.intersection(a_set) → {'c', 'd'}
```

method:difference op: -



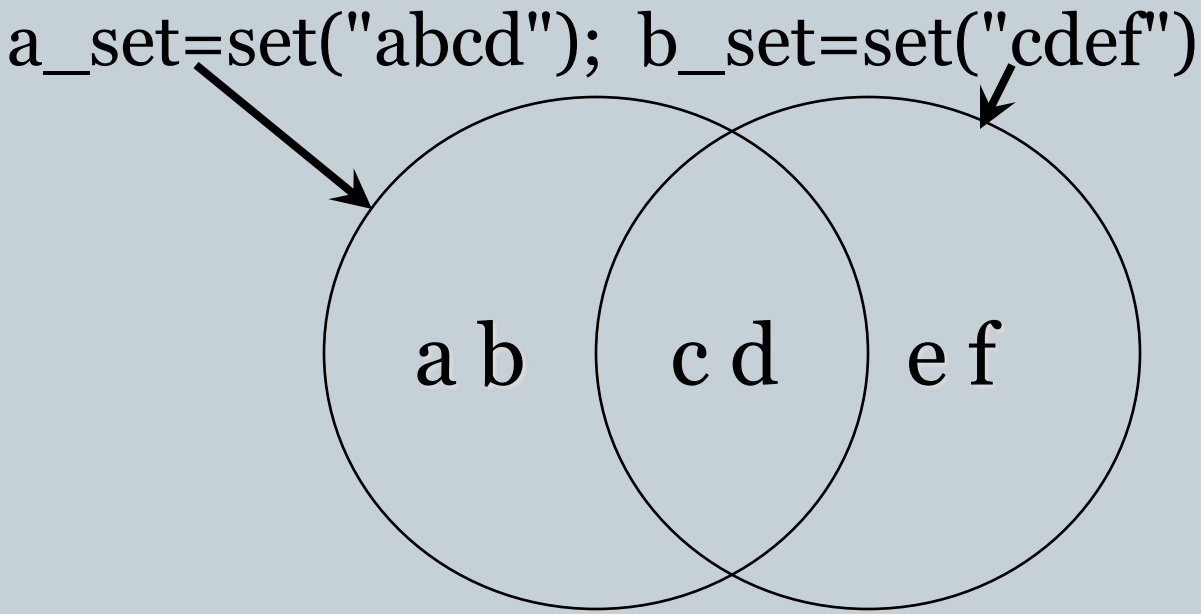

```
a_set - b_set → {'a', 'b'}
b_set.difference(a_set) → {'e', 'f'}
```

method: union, op: |




```
a_set | b_set → {'a', 'b', 'c', 'd', 'e', 'f'}
b_set.union(a_set) → {'a', 'b', 'c', 'd', 'e', 'f'}
```

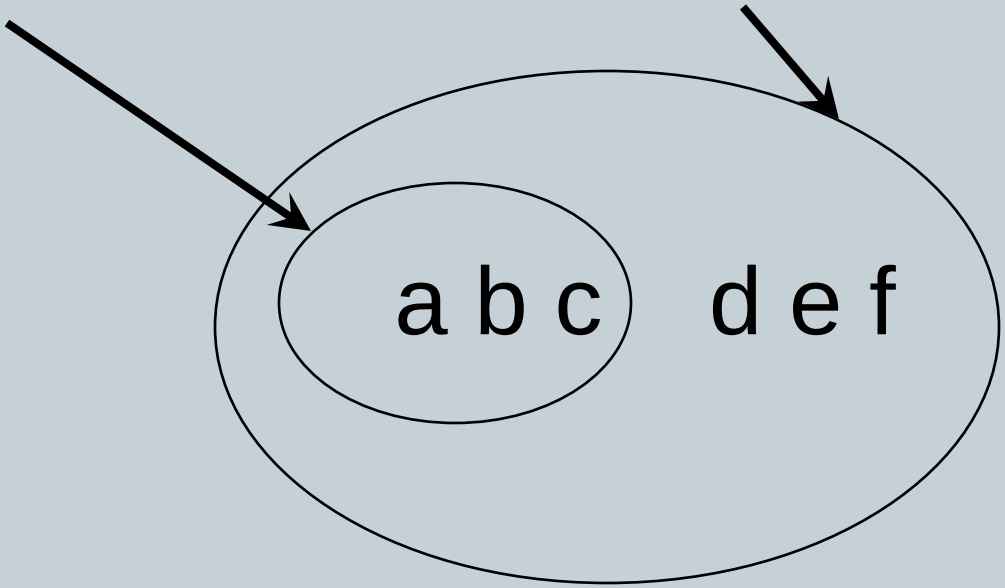
method:symmetric_difference, op: ^




```
a_set ^ b_set → {'a', 'b', 'e', 'f'}
b_set.symmetric_difference(a_set) → {'a', 'b', 'e', 'f'}
```

method: issubset, op: <=
method: issuperset, op: >=

small_set=set("abc"); big_set=set("abcdef")



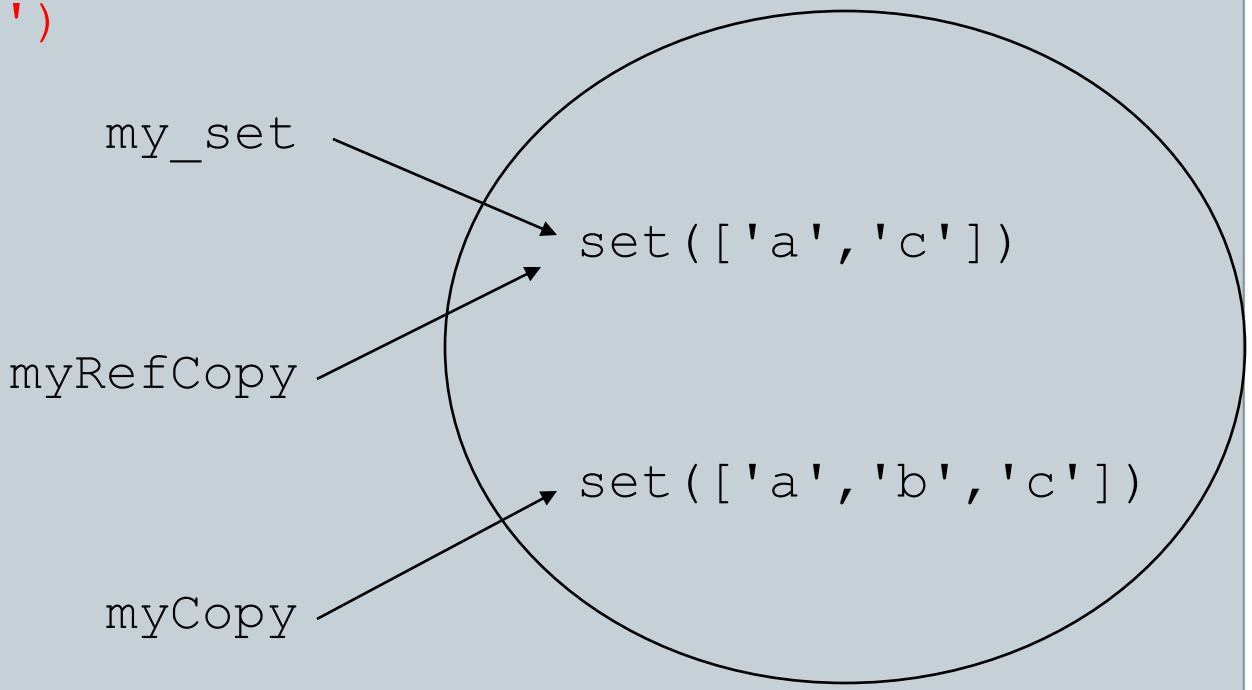

```
small_set <= big_set → True
big_set >= small_set → True
```

Other Set Ops

- `my_set.add("g")`
  - adds to the set, no effect if item is in set already
- `my_set.clear()`
  - empties the set
- `my_set.remove("g")` versus `my_set.discard("g")`
  - `remove` throws an error if "`g`" isn't there. `discard` doesn't care. Both remove "`g`" from the set
- `my_set.copy()`
  - returns a copy of `my_set`

Copy vs. assignment

```
my_set=set {'a', 'b', 'c'}
my_copy=my_set.copy()
my_ref_copy=my_set
my_set.remove('b')
```



Summary

- Dictionaries
  - Creating
  - Accessing
  - Common Operations
- Sets
  - Creating
  - Common Operations