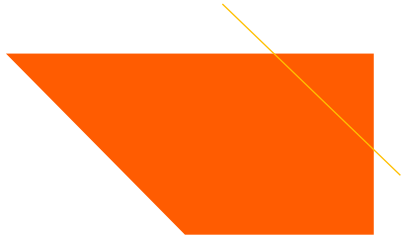
Introduction to Problem Solving in Python

COSI 10A

Class objectives

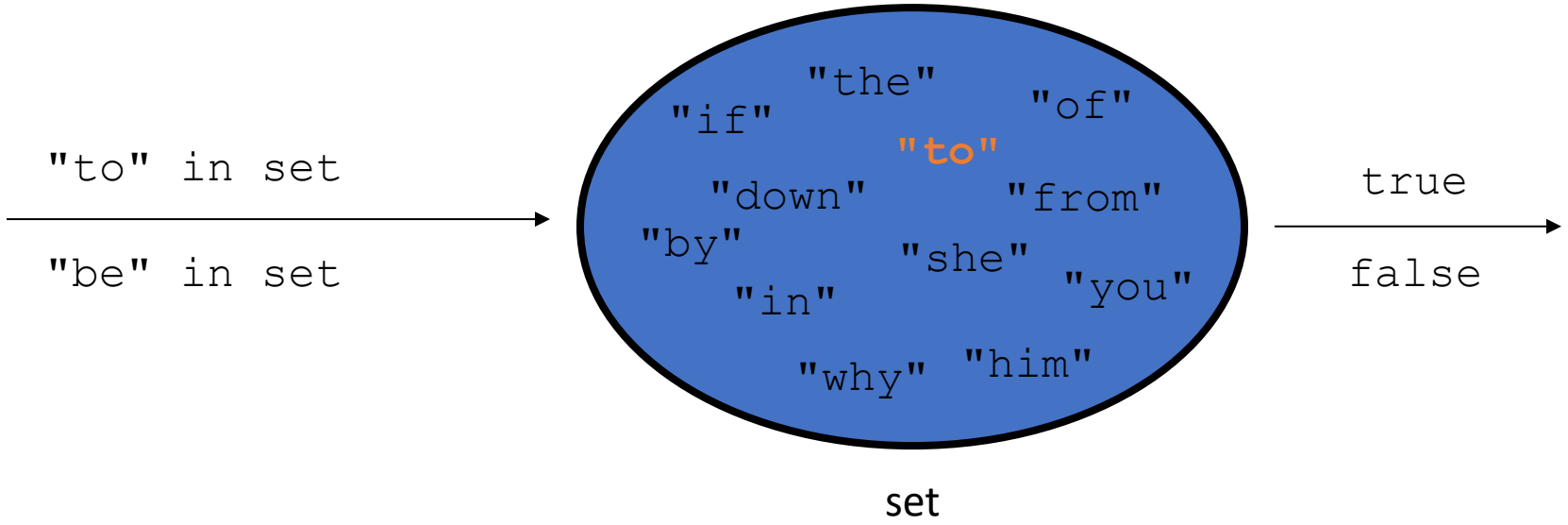
- Advanced Dictionary Usage (8.2)



Dictionaries & Sets (II)

Review: Sets

- **set**: A collection of unique values (no duplicates allowed) that can perform the following operations efficiently:
  - add, remove, search (contains)
  - We don't think of a set as having indexes; we just add things to the set in general and don't worry about order



Review: Creating a `Set`

- An empty set:

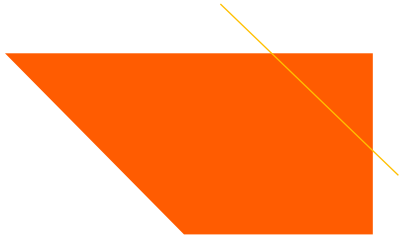
```
a = set()
```

- A set with elements in it:

```
b = {"the", "hello", "happy"}
```

`a.add(`**val**`)`	adds element val to a
`a.discard(`**val**`)`	removes val from a if present
`a.pop()`	removes and returns a random element from a
`a - b`	returns a new set containing values in a but not in b
`a | b`	returns a new set containing values in either a or b
`a & b`	returns a new set containing values in both a and b
`a ^ b`	returns a new set containing values in a or b but not both

You can also use `in`, `len()`, etc.



Exercise

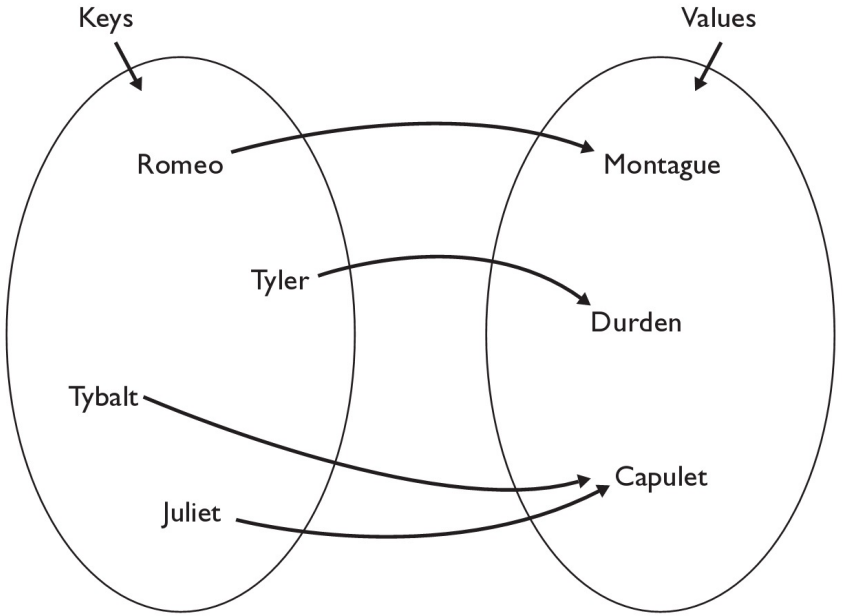
- Write a program to count the number of occurrences of each unique word in a large text file (e.g. *Moby Dick*).
  - Allow the user to type a word and report how many times that word appeared in the book.
  - Report all words that appeared in the book at least 500 times.

- What structure is appropriate for this problem?

Dictionaries

- **dictionary**: Holds a set of unique *keys* and a collection of *values*, where each key is associated with one value.
  - a.k.a. "map", "associative array", "hash"

- basic dictionary operations:
  - **put**(*key*, *value*): Adds a mapping from a key to a value.
  - **get**(*key*): Retrieves the value mapped to the key.
  - **remove**(*key*): Removes the given key and its mapped value.

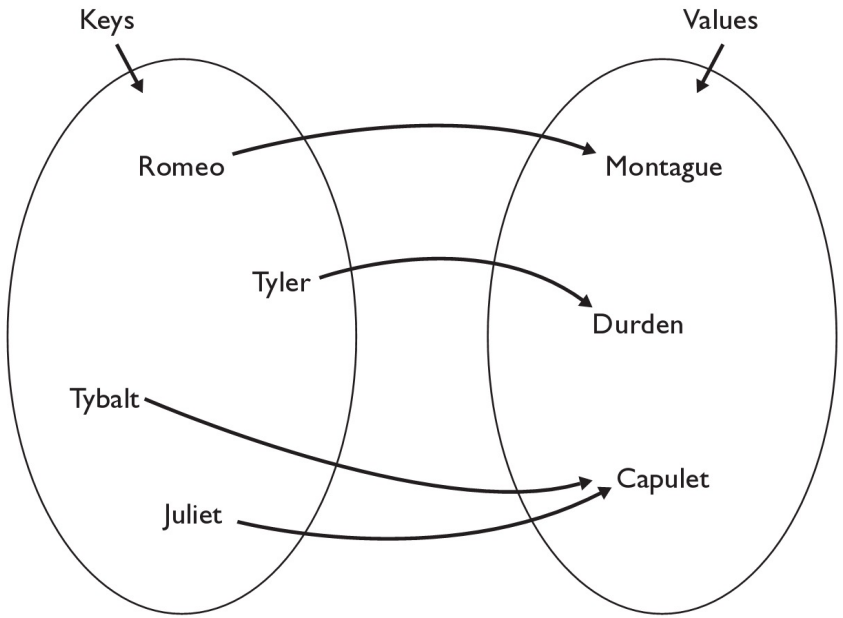


`my_dict["Juliet"]` returns `"Capulet"`

Creating dictionaries

- Creating a dictionary
  - {**key** : **value, …, key : value**}

```
my_dict = {"Romeo": "Montague",
           "Tyler" :"Durden",
           "Tybalt" : "Capulet",
           "Juliet" :"Capulet" }
```



my_dict[**key**] = **value**

adds a mapping from the given key to the given value;
if the key already exists, replaces its value with the given one

Accessing values:

- my_dict[**key**]

  returns the value mapped to the given key (error if key not found)

`my_dict["Juliet"]` produces `"Capulet"`

Dictionary functions

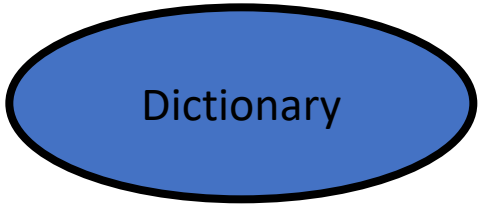
`my_dict[`**key**`] = `**value**	adds a mapping from the given key to the given value; if the key already exists, replaces its value with the given one
`my_dict[`**key**`]`	returns the value mapped to the given key (error if key not found)
`items()`	return a new view of the dictionary's items ((key, value) pairs)
`pop(`**key**`)`	removes any existing mapping for the given key and returns it (error if key not found)
`popitem()`	removes and returns an arbitrary (key, value) pair (error if empty)
`keys()`	returns the dictionary's keys
`values()`	returns the dictionary's values

You can also use `in`, `len()`, etc.

Using dictionaries

- A dictionary allows you to get from one half of a pair to the other.
  - Remembers one piece of information about every index (key).

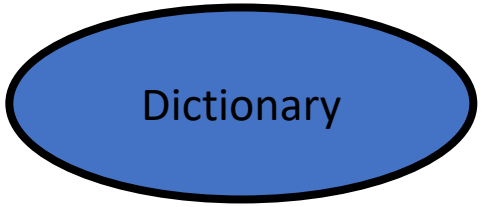
```
#          key       value
my_dict["Suzy"] = "206-685-2181"
```



  - Later, we can supply only the key and get back the related value:

    Allows us to ask: *What is Suzy's phone number?*

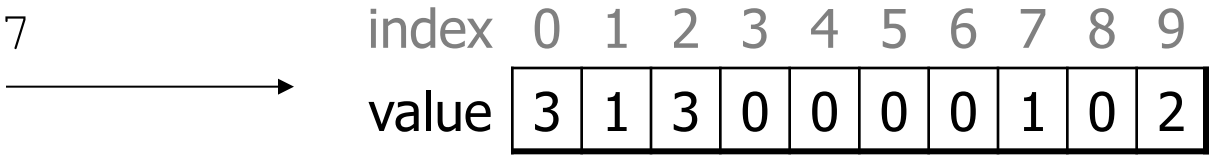
```
my_dict["Suzy"]
```




```
"206-685-2181"
```

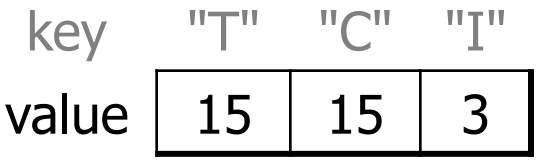
Dictionaries and tallying

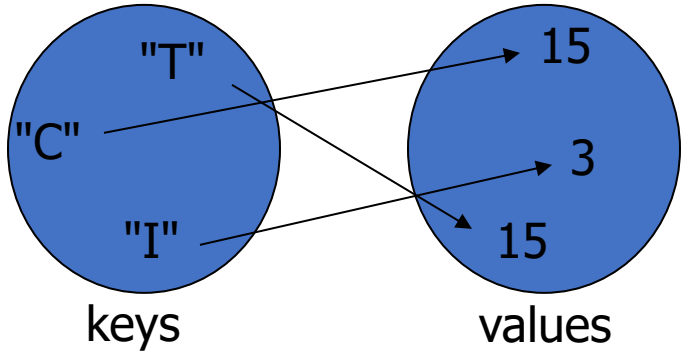
- a dictionary can be thought of as generalization of a tallying list
  - the "index" (key) doesn't have to be an `int`
  - count digits: `22092310907`




```
# (T)rump, (C)linton, (I)ndependent
```

  - count votes: `"TCCCCCCTTTTTCCCCCCTCTTITCTTITCCTIC"`





Looping over a set or dictionary?

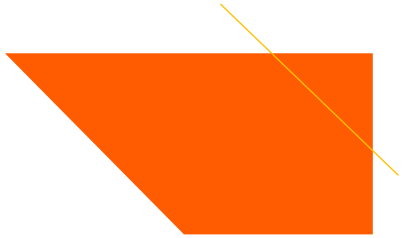
- You must use a `for element in` structure loop
  - needed because sets have no indexes; can't `get` element `i`

Example:

```
for item in a:
    print(item)
```

Outputs:

```
the
happy
hello
```

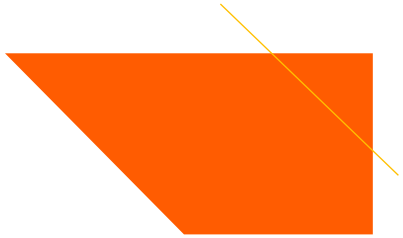


items, keys and values

- `items` function returns tuples of each key-value pair
  - can loop over the keys in a for loop

```
ages = {}
ages["Merlin"] = 4
ages("Chester"] = 2
ages["Percival"] = 12
for cat, age in ages.items()):
    print(name + " -> " + str(age))
```

- `values` function returns all values in the dictionary
  - no easy way to get from a value to its associated key(s)
- `keys` function returns all keys in the dictionary



Exercise

- Write a program to count the number of occurrences of each unique word in a large text file (e.g. *Moby Dick*).
  - Allow the user to type a word and report how many times that word appeared in the book.
  - Report all words that appeared in the book at least 500 times.

- What structure is appropriate for this problem? Dictionaries

Exercise

Consider the following function:

```
def mystery(alls, letters):
    for i in range(len(alls)):
        if alls[i] in letters.values():
            if alls[i] not in letters.keys():
                letters[alls[i]] = i
            else:
                letters[alls[i]] = i + 1
```

What is in the dictionary after calls with the following parameters?

alls: [b, l, u, e] letters:{s:b, p:t, o:u, t:t}

dictionary:_____

alls:[k, e, e, p] letters: {s:y, a:k, f:e, e:f}

dictionary:_____

alls:[s, o, b, e, r] letters: {b:b, o:o, o:o, k:k, s:s}

dictionary:_____

Exercise

Consider the following function:

```
def mystery(alls, letters):
    for i in range(len(alls)):
        if alls[i] in letters.values():
            if alls[i] not in letters.keys():
                letters[alls[i]] = i
            else:
                letters[alls[i]] = i + 1
```

What is in the dictionary after calls with the following parameters?

all: [b, l, u, e] letters:{s:b, p:t, o:u, t:t}

dictionary: _{'o': 'u', 'p': 't', 'b': 0, 's': 'b', 'u': 2, 't': 't'} __

all:[k, e, e, p] letters: {s:y, a:k, f:e, e:f}

dictionary:_ {'s': 'y', 'a': 'k', 'f': 'e', 'e': 3, 'k': 0} ___

all:[s, o, b, e, r] letters: {b:b, o:o, o:o, k:k, s:s}

dictionary:__ {'o': 2, 'k': 'k', 's': 1, 'b': 3} __

Choosing the Right Structure

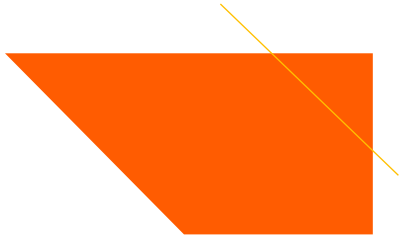
What is the right structure?

- You want to store a bunch of colors so you can later choose one at random.
- Students’ names and their grades on a project.
- Friends' names and their phone numbers
- Height, width and location of a sports field.
- Movies a person has watched.
- Items in a shopping cart.
- A student's grades.

What is the right structure?

- The grades for all students in a class
- All books in a store arranged by category
- Many recipes each containing many steps
- Phone numbers that have been called this month on a phone plan divided by area and country code for billing simplicity

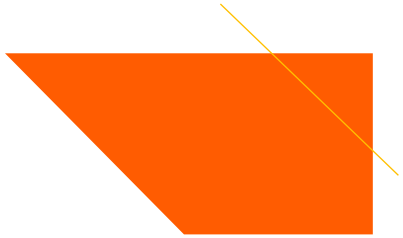
2D Structures



Exercise

- We would like to store data for the class so that we can:
  - Access the entire class list easily
  - Access a section list easily

- What structure is appropriate for this problem?
  - Sometimes it can be helpful to store a structure inside another structure



2d Structure Access

- Given the following structure:

```
grades = {'Ali':[10, 16, 20, 13, 3, 17],
          'Ken':[9, 16, 8, 19, 20, 20],
          'Daniel':[8, 10, 20, 20, 20, 20]}
```

- How can I access Ken's grade on project 3?
- How can I find out how many students are in my class / grades?
- How can I find out how many projects a student has done?