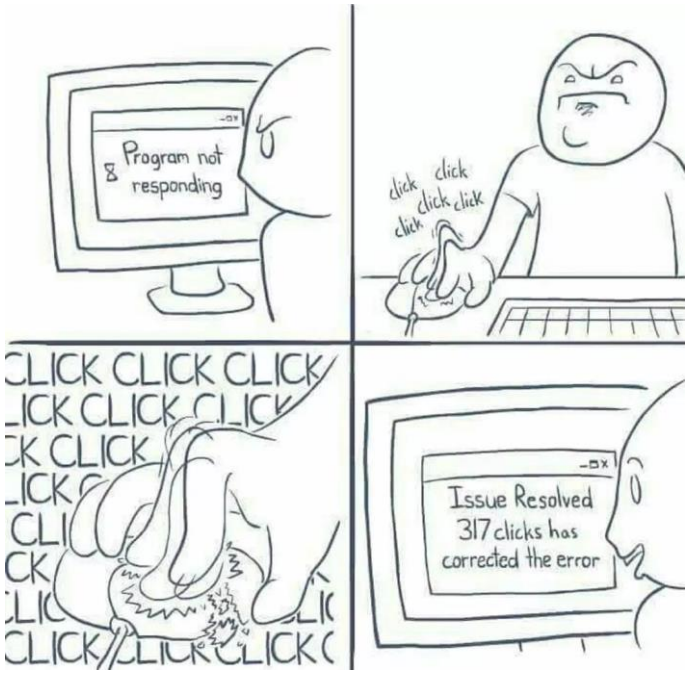
CSc 110, Autumn 2017

Lecture 30: Sets and Dictionaries

Adapted from slides by Marty Stepp and Stuart Reges



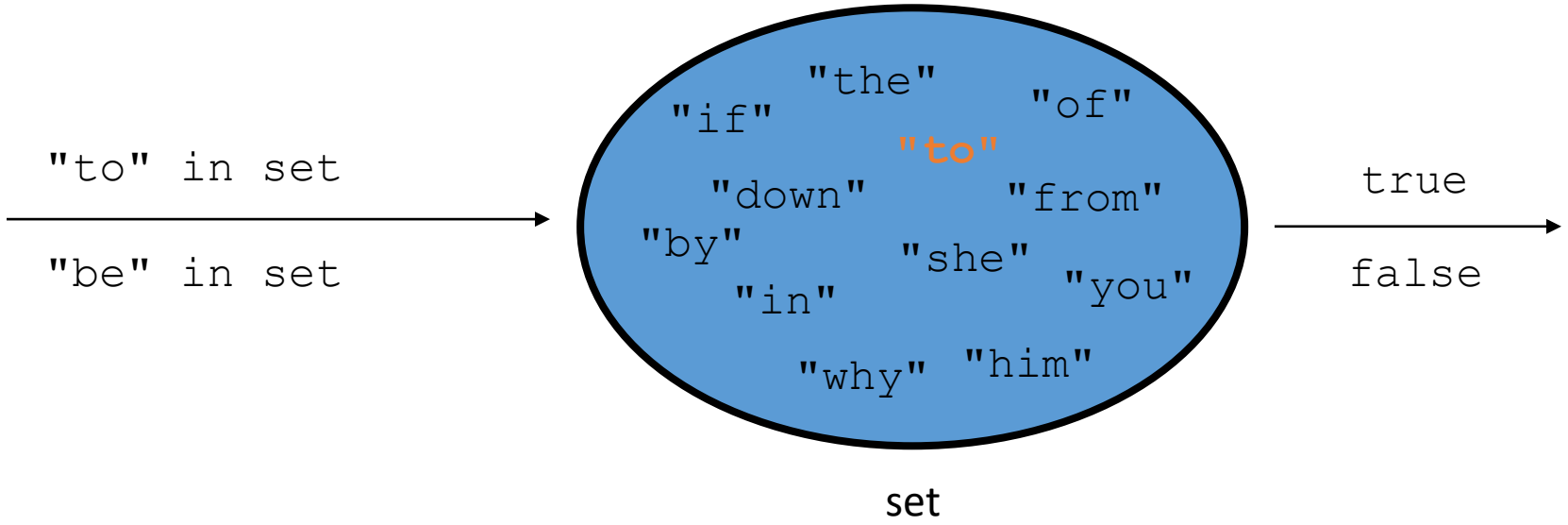
Exercise

- Write a program that counts the number of unique words in a large text file (say, *Moby Dick* or the King James Bible).
  - Store the words in a structure and report the # of unique words.
  - Once you've created this structure, allow the user to search it to see whether various words appear in the text file.

- What structure is appropriate for this problem? List? Tuple?

Sets

- **set**: A collection of unique values (no duplicates allowed)
- Sets are not indexed. They do not have an order.
- The following operations can be performed efficiently on sets:
  - add, remove, search (contains)



Creating a set

- Use the function `set`:

```
a = set()
```

- Use {**value, …, value**}:

```
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.

Looping over a set?

- 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
```

Exercise

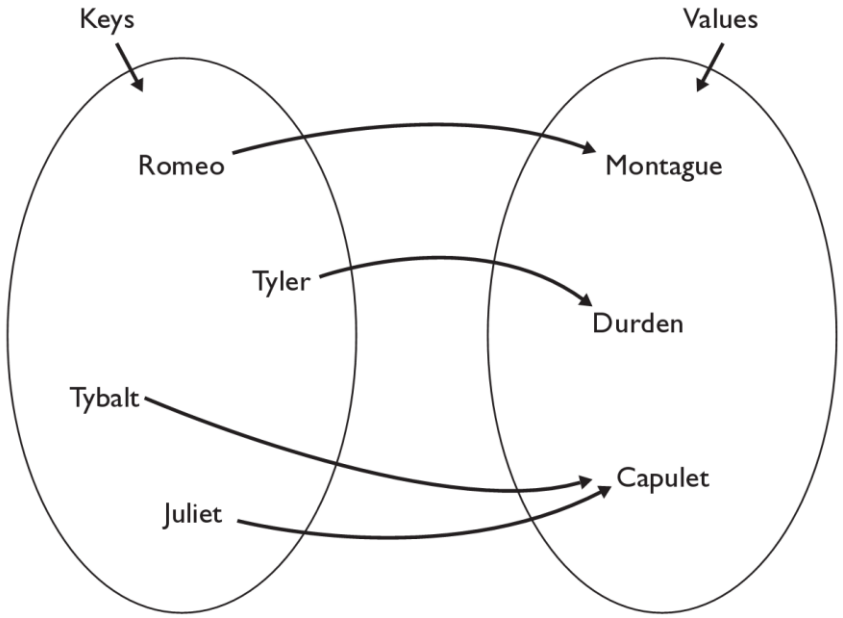
- Write a program to count the number of occurrences of each unique word in a large text file (e.g. *Moby Dick*).
  - Allow the user to type a word and report how many times that word appeared in the book.
  - Report all words that appeared in the book at least 500 times.

- What structure is appropriate for this problem?

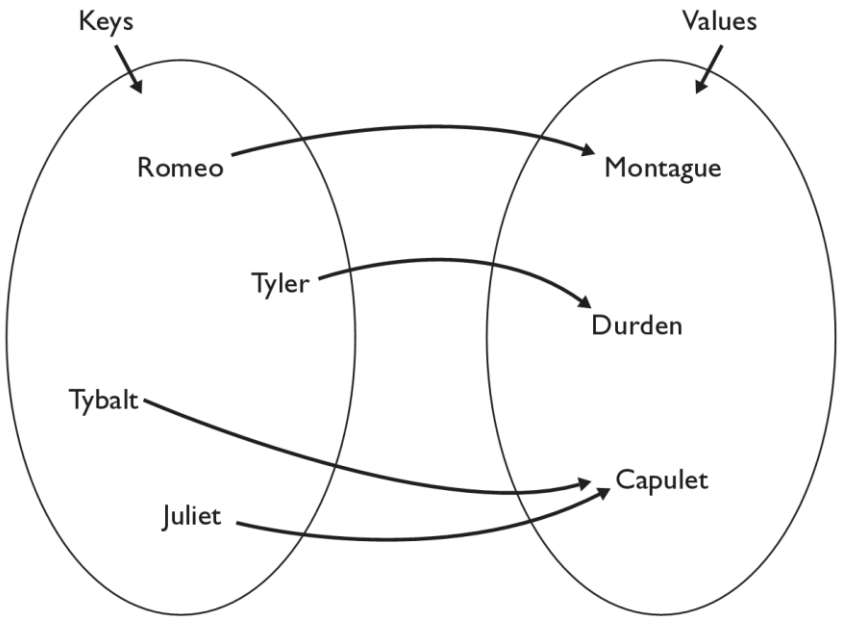
Dictionaries

- **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:
  - Add a mapping from a key to a value.
  - Retrieve a value mapped to a key.
  - Remove a given key and its mapped value.



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"`

Dictionaries and tallying

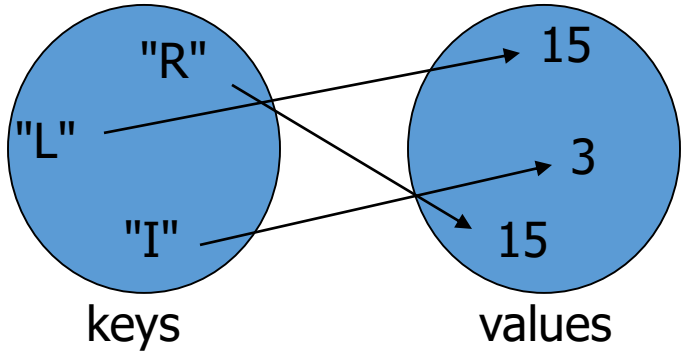
- a dictionary can be thought of as generalization of a tallying list
  - the "index" (key) doesn't have to be an `int`
  - count digits: `2209231090`7

index	0	1	2	3	4	5	6	7	8	9
value	3	1	3	0	0	0	0	1	0	2

```
# (Roosevelt), (L)andon, (I)ndependent
```

  - count votes: "`RLLLLLLRRRRRLLLLLLRLRRIRLRRIRLLRIR`"

key	"R"	"L"	"I"
value	15	15	3



Dictionary operations

`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.

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(cat + " -> " + 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

Consider the following function:

```
def mystery(list1, list2):
    result = {}
    for i in range(0, len(list1)):
        result[list1[i]] = list2[i]
        result[list2[i]] = list1[i]
    return result
```

What is returned after calls with the following parameters?

`list1: [b, l, u, e]` `list2:[s, p, o, t]`

dictionary returned:_____

`list1:[k, e, e, p]` `list2: [s, a, f, e]`

dictionary returned:_____

`list1:[s, o, b, e, r]` `list2: [b, o, o, k, s]`

dictionary returned:_____