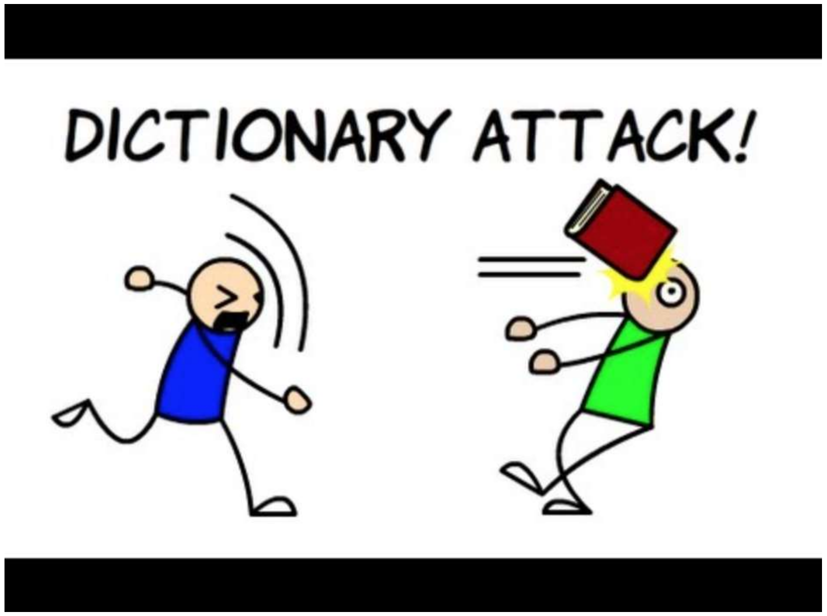
CSc 110, Spring 2017

Lecture 29: 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?

Unique Words

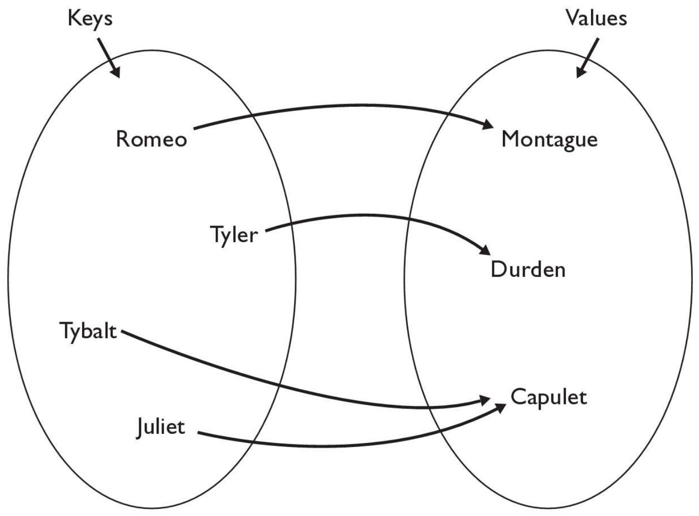
```
# outputs the number unique words in a file
def main():
    all_words = file_to_words("mobydick.txt")
    print("unique word count " + str(len(all_words)))

# creates and returns a set containing all of the words from the
# file with the passed in name stripped of punctuation.
def file_to_words(file_name):
    file = open(file_name)
    words = file.read()
    # get rid of punctuation
    words = words.replace(",", "")
    words = words.replace(".", "")
    words = words.replace("!", "")
    words = words.split()
    s = set()
    for word in words:
        s.add(word.lower())
    return s
main()
```

Exercise

- Write a program to <u>count the number of occurrences</u> 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

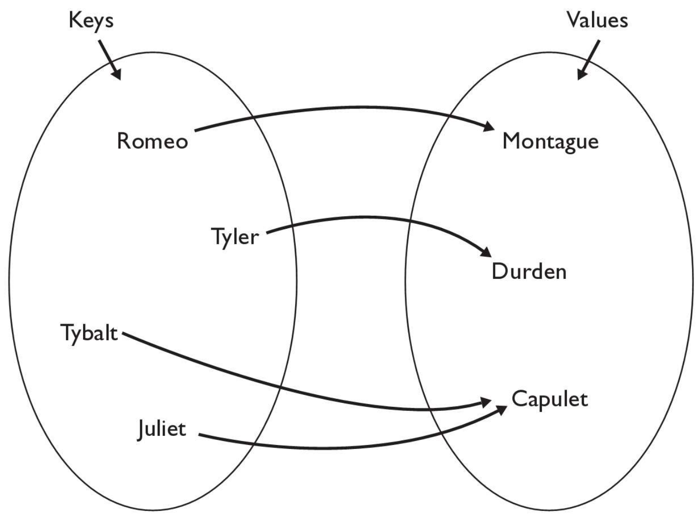
- **dictionary**: Holds a collection of zero or more *key/value* pairs
  - a.k.a. "map", "associative array", "hash"
- basic dictionary operations:
  - Add a mapping from a key to a value.
  - Retrieve a value mapped to a key.
  - Remove a given key and its mapped value.



Creating dictionaries

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

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



dictionary[**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:

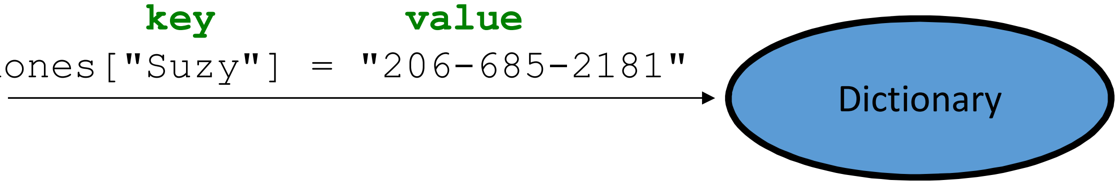
- **dictionary**[**key**]
  retuns the value mapped to the given key (error if key not found)

`names["Juliet"]` **produces** `"Capulet"`

Using dictionaries

- A dictionary allows you to get from one half of a pair to the other.
  - Associates one piece of information for every key.

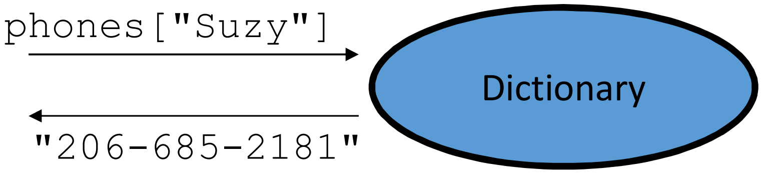
```
#          key          value
 phones["Suzy"] = "206-685-2181"
```



  - Using the key as an index produces the related value:

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

```
phones["Suzy"]

 "206-685-2181"
```



- Lists must be indexed by integers
  - count digits: `22092310907`

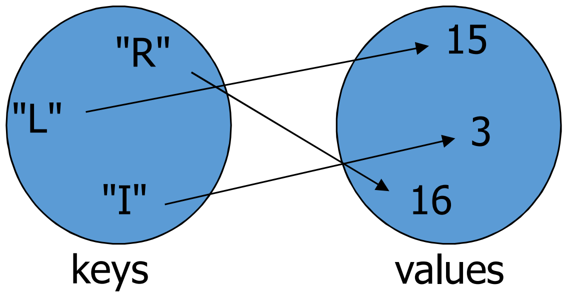
index	0	1	2	3	4	5	6	7	8	9
value	3	1	3	0	0	0	0	1	0	2

- Dictionaries can be indexed by integers, strings, tuples and more

```
# (R)oosevelt, (L)andon, (I)ndependent
```

  - count votes: `"RLLLLLLRRRRRLLLLLLRLRRIRLRRIRLLRIRR"`

key	"R"	"L"	"I"
value	16	15	3



Checking for a key with the `in` operator

- The `in` operator returns `True` if the dictionary contains the specified key and `False` otherwise.

```
>>> ages = {}
>>> ages["Joe"] = 10
>>> ages
{'Joe': 10}
>>> "Joe" in ages
True
>>> "Tom" in ages
False
```

Looping through dictionaries

- The `for` loop can be used to loop through the keys in a dictionary

```
ages = {}
ages["Merlin"] = 4
ages["Chester"] = 2
ages["Percival"] = 12
for name in ages:
    print(name, ages[name])
```

Output:

```
Merlin 4
Chester 2
Percival 12
```

Example

- Write a function `count_chars` that takes a string and returns a dictionary of the counts of all characters in the string.

  Using a dictionary:

    The keys will be the characters

    The values will be the counts.

Example

- Write a function `count_chars` that takes a string and returns a dictionary of the counts of all characters in the string.

Using the name `counts` for the dictionary, to update the count, we use the following:

```
counts[c] = counts[c] + 1
```

What happens the first time we encounter a new character `c`?

Must check first to see if it's there.

Dictionary methods

`items()`	returns a sequence of tuples (key, value) representing the 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 `len()`, etc.

`items`, `keys` and `values`

- The `items` method can be used to loop through all the key/value pairs in a dictionary

```
ages = {}
ages["Merlin"] = 4
ages["Chester"] = 2
ages["Percival"] = 12
for tup in ages.items():
    print(tup[0] + " -> " + str(tup[1]))
```

- `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