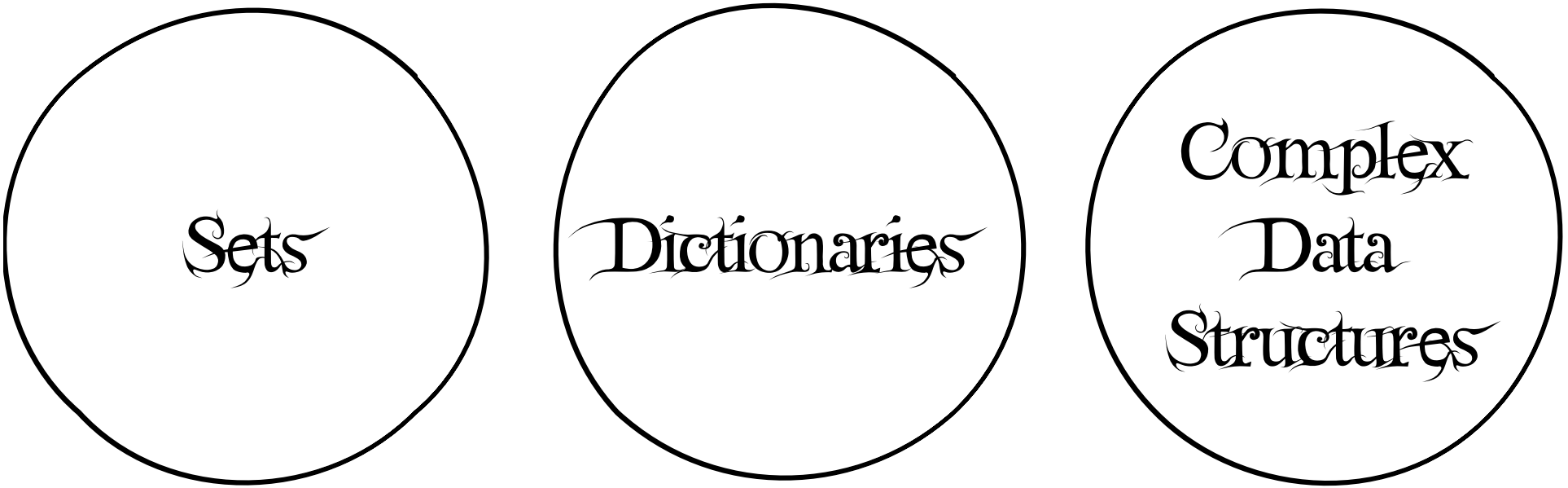
Chapter 8

SETS AND DICTIONARIES

Chapter Contents



Chapter Goals

- To build and use a set container
- To learn common set operations for processing data
- To build and use a dictionary container
- To work with a dictionary for table lookups
- To work with complex data structures

In this chapter, we will learn how to work with two more types of containers (sets and dictionaries) as well as how to combine containers to model complex structures.

Sets

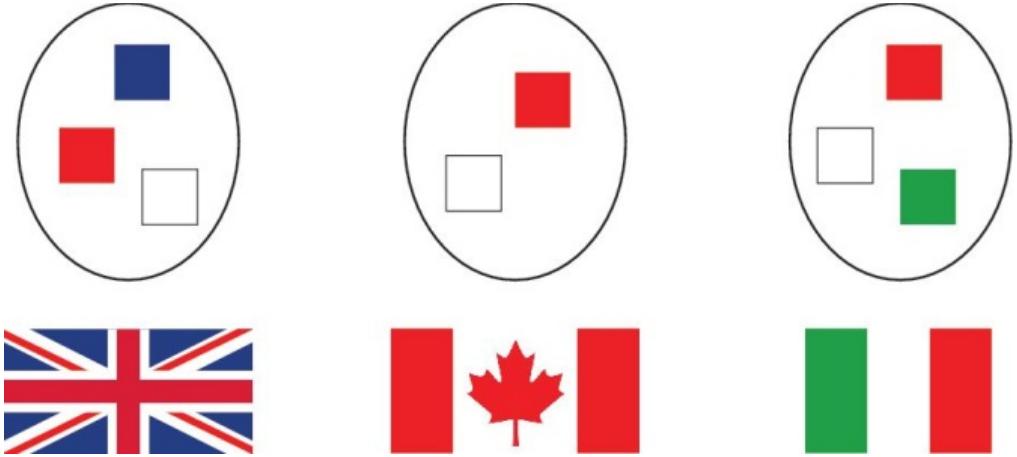
SECTION 8.1

Sets

- A set is a container that stores a collection of unique values
- Unlike a list, the elements or members of the set are not stored in any particular order and cannot be accessed by position
- Operations are the same as the operations performed on sets in mathematics
- Because sets do not need to maintain a particular order, set operations are much faster than the equivalent list operations

Example Set

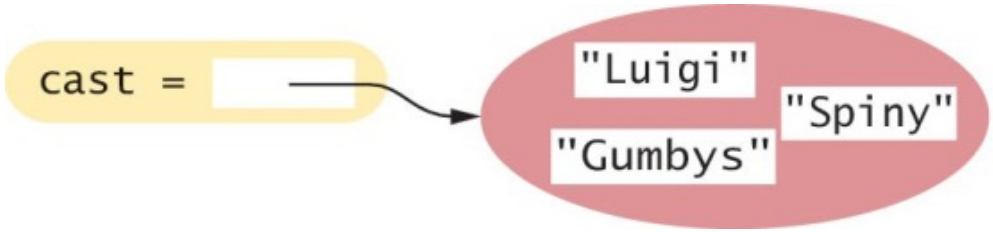
- This set contains three sets of colors—the colors of the British, Canadian, and Italian flags
- In each set, the order does not matter, and the colors are not duplicated in any one of the sets



Creating and Using Sets

- To create a set with initial elements, you can specify the elements enclosed in braces, just like in mathematics:

```
cast = { "Luigi", "Gumbys", "Spiny" }
```



- Alternatively, you can use the set() function to convert any sequence into a set:

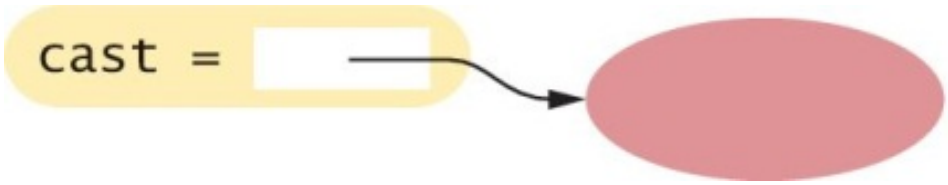
```
names = ["Luigi", "Gumbys", "Spiny"]
cast = set(names)
```

Creating an Empty Set

- For historical reasons, you cannot use { } to make an empty set in Python
- Instead, use the `set()` function with no arguments:

```
cast = set()
```

- As with any container, you can use the `len()` function to obtain the number of elements in a set:




```
numberOfCharacters = len(cast)   # In this case it's zero
```

Set Membership: `in`

- To determine whether an element is contained in the set, use the `in` operator or its inverse, the `not in` operator:

```
if "Luigi" in cast :
    print("Luigi is a character in Monty Python's Flying Circus.")
else :
    print("Luigi is not a character in the show.")
```

Accessing Set Elements

- Because sets are unordered, you cannot access the elements of a set by position as you can with a list
- We use a for loop to iterate over the individual elements:

```
print("The cast of characters includes:")
for character in cast :
    print(character)
```

- Note that the order in which the elements of the set are visited depends on how they are stored internally

Accessing Elements (2)

- For example, the previous loop above displays the following:

```
The cast of characters includes:
Gumbys
Spiny
Luigi
```

- Note that the order of the elements in the output is different from the order in which the set was created

Displaying Sets In Sorted Order

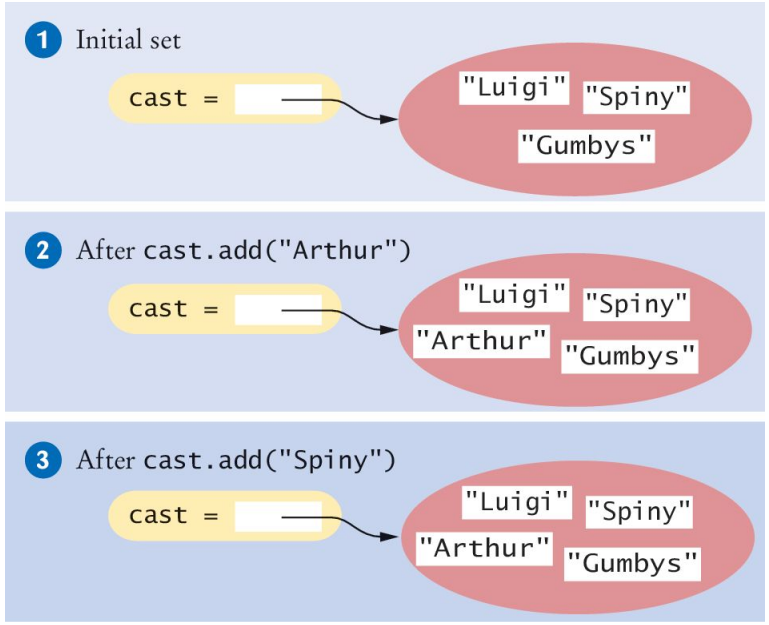
- Use the `sorted()` function, which returns a list ***(not a set)*** of the elements in sorted order
- The following loop prints the cast in sorted order:

```
for actor in sorted(cast) :
    print(actor)
```

Adding Elements

- Sets are ***mutable*** collections, so you can add elements by using the `add()` method:

```
cast = set(["Luigi", "Gumbys", "Spiny"])   #1
cast.add("Arthur")                         #2
cast.add("Spiny")                          #3
```



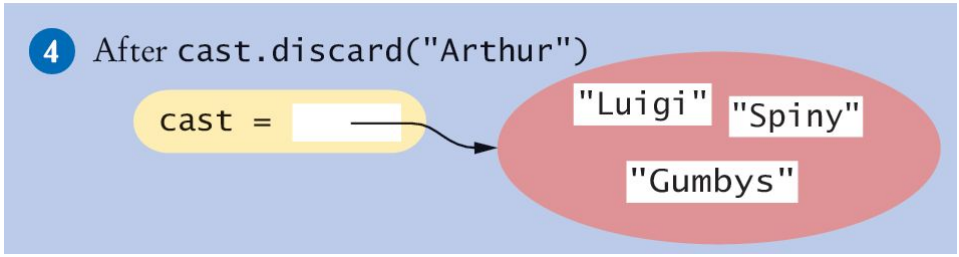
Arthur is not in the set, so it is added to the set and the size of the set is increased by one

Spiny is already in the set, so there is no effect on the set

Removing Elements: `discard()`

- The `discard()` method removes an element if the element exists:

```
cast.discard("Arthur")  #4
```



- It has no effect if the given element is not a member of the set:

```
cast.discard("The Colonel")   # Has no effect
```

Removing Elements: `remove()`

- The `remove()` method, on the other hand, removes an element if it exists, but raises an exception if the given element is not a member of the set:

```
cast.remove("The Colonel")   # Raises an exception
```

- For this class we will use the discard() method

Removing Elements: `clear()`

- Finally, the `clear()` method removes *all* elements of a set, leaving the empty set:

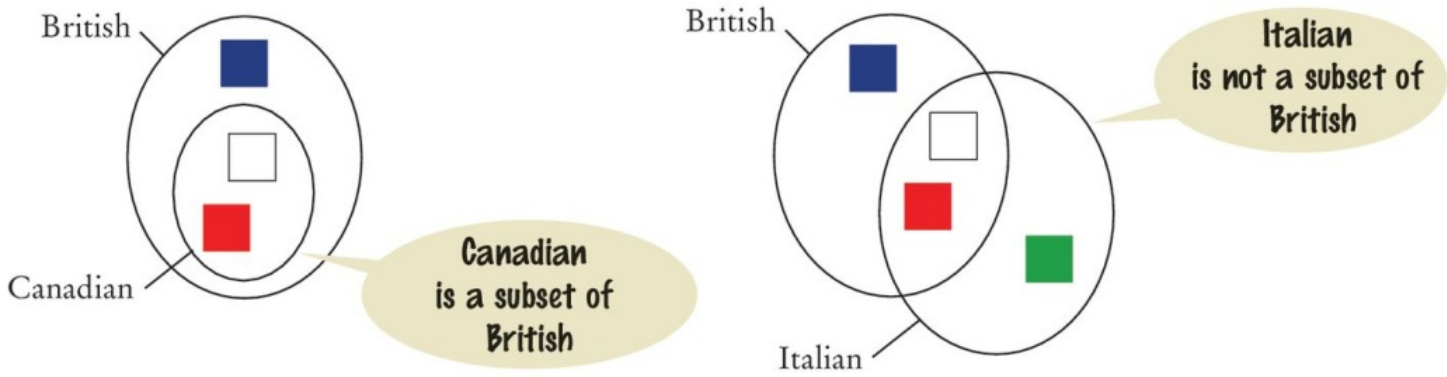
```
cast.clear() # cast now has size 0
```

Example

```
def functionC(mySetC):
    print("Starting functionC")
    mySetC.add("Hunter")
    mySetC.remove("Iba")
    mySetC.remove("Iba")
    print("Here is what mySet looks like in functionC:", mySetC)
    print("Ending functionC")

def functionB(mySetB):
    print("Starting functionB")
    functionC(mySetB)
    print("Here is what mySet looks like in functionB:", mySetB)
    print("Ending functionB")


def functionA(mySetA):
    try:
        print("Starting functionA")
        functionB(mySetA)
        print("Here is what mySet looks like in functionA:", mySetA)
        print("Ending functionA")
    finally:
        print("Ending main")



def main():
    mySet = {"Patterson", "Rosentrater", "Iba", "Hunter"}
    try:
        print("Starting main")
        functionA(mySet)
        print("Here is what mySet looks like in main:", mySet)
    except Exception as myException :
        print("I caught an exception in main:",str(myException))
    finally:
        print("Ending main")

main()
```

Subsets

- A set is a *subset* of another set *if and only if* every element of the first set is also an element of the second set
- In the image below, the Canadian flag colors are a subset of the British colors
- The Italian flag colors are not.



The `issubset()` Method

- The `issubset()` method returns `True` or `False` to report whether one set is a subset of another:

```
canadian = { "Red", "White" }
british = { "Red", "Blue", "White" }
italian = { "Red", "White", "Green" }

# True
if canadian.issubset(british) :
    print("All Canadian flag colors occur in the British flag.")

# True
if not italian.issubset(british) :
    print("At least one of the colors in the Italian flag does
       not.")
```

Set Equality / Inequality

- We test set equality with the “==“ and “!=“ operators
- Two sets are equal ***if and only if*** they have exactly the same elements

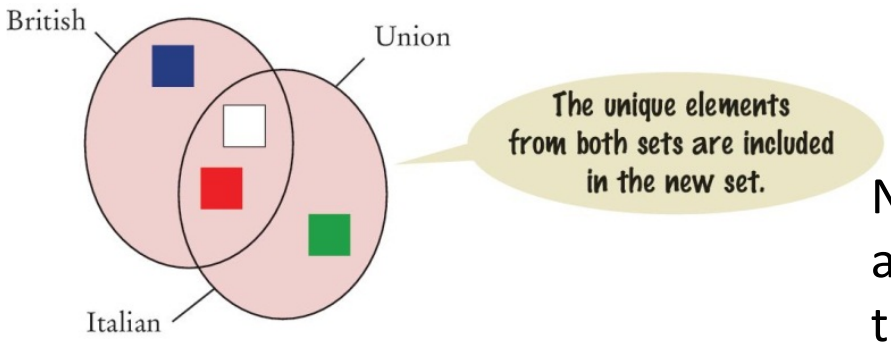
```
french = { "Red", "White", "Blue" }
if british == french :
    print("The British and French flags use the same colors.")
```

Set Union: `union()`

- The *`union`* of two sets contains all of the elements from both sets, with duplicates removed

```
# inEither: The set {"Blue", "Green", "White", "Red"}
inEither = british.union(italian)
```

- Both the British and Italian sets contain the colors Red and White, but the union is a set and therefore contains only one instance of each color

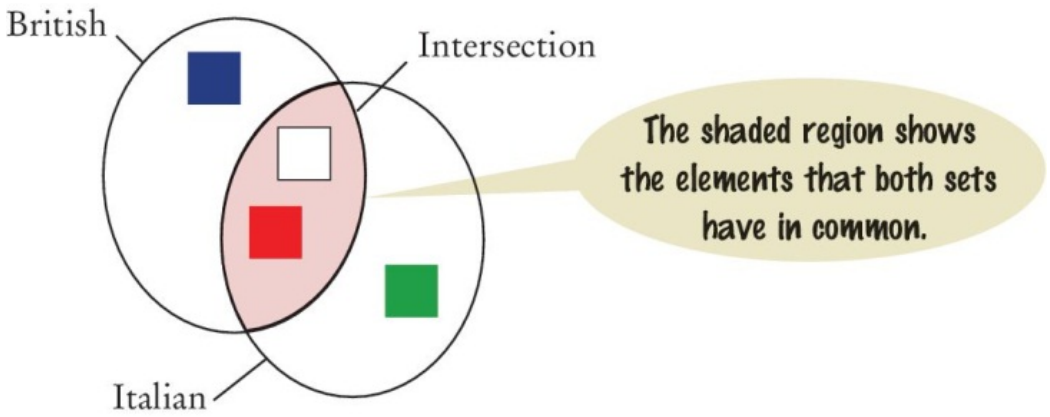


Note that the union() method returns a new set. It does not modify either of the sets in the call

Set Intersection: `intersection()`

- The *`intersection`* of two sets contains all of the elements that are in *both* sets

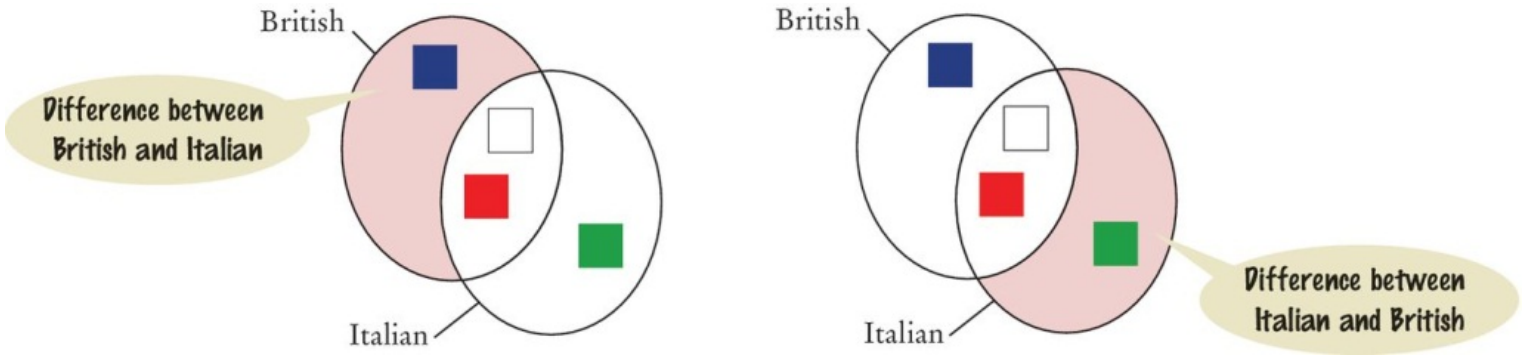
```
# inBoth: The set {"White", "Red"}
inBoth = british.intersection(italian)
```



Difference of Two Sets: `difference()`

- The *`difference`* of two sets results in a new set that contains those elements in the first set that are not in the second set

```
print("Colors that are in the Italian flag but not the
   British:")
print(italian.difference(british)) # Prints {'Green'}
```



Common Set Operations

Table 1 Common Set Operations

Operation	Description

`s = set()`
`s = set(`*seq*`)`
`s = {`e_1`, `e_2`, ..., `e_n`}`
	Creates a new set that is either empty, a duplicate copy of sequence *seq*, or that contains the initial elements provided.

`len(`*s*`)`	Returns the number of elements in set *s*.
element `in` *s* *element* `not in` *s*	Determines if *element* is in the set.
s`.add(`*element*`)`	Adds a new element to the set. If the element is already in the set, no action is taken.
s`.discard(`*element*`)` *s*`.remove(`*element*`)`	Removes an element from the set. If the element is not a member of the set, `discard` has no effect, but `remove` will raise an exception.
s`.clear()`	Removes all elements from a set.
s`.issubset(`*t*`)`	Returns a Boolean indicating whether set *s* is a subset of set *t*.

Common Set Operations (2)

Table 1 Common Set Operations	
`s == t` `s != t`	Returns a Boolean indicating whether set *s* is equal to set *t*.
`s.union(t)`	Returns a new set that contains all elements in set *s* and set *t*.
`s.intersection(t)`	Returns a new set that contains elements that are in *both* set *s* and set *t*.
`s.difference(t)`	Returns a new set that contains elements in *s* that are not in set *t*.

Remember: `union`, `intersection` and `difference` return new sets
They do not modify the set they are applied to

Simple Examples

- Open the file: set examples.py

Set Example: Spell Checking

- The program spellcheck.py reads a file that contains correctly spelled words and places the words in a set
- It then reads all words from a document—here, the book *Alice in Wonderland*—into a second set
- Finally, it prints all words from the document that are not in the set of correctly spelled words
- Open the file spellcheck.py

Example: `Spellcheck.py`

```
1  ##
2  #  This program checks which words in a file are not present in a list of
3  #  correctly spelled words.
4  #
5
6  #  Import the split function from the regular expression module.
7  from re import split
8
9  def main() :
10    # Read the word list and the document.
11    correctlySpelledWords = readWords("words")
12    documentWords = readWords("alice30.txt")
13
14    # Print all words that are in the document but not the word list.
15    misspellings = documentWords.difference(correctlySpelledWords)
16    for word in sorted(misspellings) :
17       print(word)
```

Example: `Spellcheck.py`

```
24  def readWords(filename) :
25     wordSet = set()
26     inputFile = open(filename, "r")
27
28     for line in inputFile :
29        line = line.strip()
30        # Use any character other than a-z or A-Z as word delimiters.
31        parts = split("[^a-zA-Z]+", line)
32        for word in parts :
33           if len(word) > 0 :
34              wordSet.add(word.lower())
35
36     inputFile.close()
37     return wordSet
38
39  # Start the program.
40  main()
```

Execution: `Spellcheck.py`

```
...
champaign
chatte
clamour
comfits
conger
croqueted
croqueting
cso
daresay
dinn
dir
draggled
dutchess
...
```

Programming Tip

- When you write a program that manages a collection of unique items, sets are far more efficient than lists
- Some programmers prefer to use the familiar lists, replacing

```
itemSet.add(item)
```

with:

```
if (item not in itemList)
    itemList.append(item)
```

- However, the resulting program is much slower.
  - The speed factor difference is over 10 times

Counting Unique Words

Problem Statement

- We want to be able to count the number of unique words in a text document
  - “Mary had a little lamb” has 57 unique words
- Our task is to write a program that reads in a text document and determines the number of unique words in the document

Step One: Understand the Task

- To count the number of unique words in a text document we need to be able to determine if a word has been encountered earlier in the document
  - Only the first occurrence of a word should be counted
- The easiest way to do this is to read each word from the file and add it to the set
  - Because a set cannot contain duplicates we can use the add method
  - The add method will prevent a word that was encountered earlier from being added to the set
- After we process every word in the document the size of the set will be the number of unique words contained in the document

Step Two: Decompose the Problem

The problem can be split into several simple steps:

Create an empty set

for each word in the text document
    Add the word to the set

Number of unique words = the size of the set

- Creating the empty set, adding an element to the set, and determining the size of the set are standard set operations
- Reading the words in the file can be handled as a separate task

Step Three: Build the Set

- We need to read individual words from the file. For simplicity in our example we will use a literal file name

```
inputFile = open(“nurseryrhyme.txt”, “r”)
For line in inputFile :
    theWords = line.split()
    For words in theWords :
        Process word
```

- To count unique words we need to remove any nonletters and remove capitalization
- We will design a function to “clean” the words before we add them to the set

Step Four: Clean the Words

- To strip out all the characters that are not letters we will iterate through the string, one character at a time, and build a new “clean” word

```
def clean(string) :
  result = “”
  for char in string :
    if char.isalpha() :
      result = result + char
return result.lower()
```

Step Five: Some Assembly Required

- Implement the main() function and combine it with the other functions
- Open the file: countwords.py