The Practice of Computing Using

PYTHON

William Punch   Richard Enbody

Chapter 8

Dictionaries and Sets

Addison-Wesley
is an imprint of

PEARSON

More Data Structures

- We have seen the list and tuple data structures and their uses.
- We will now examine two, more advanced data structures: the *set* and the *dictionary.*
- In particular, the dictionary is an important, very useful part of Python as well as generally useful to solve many problems.

Dictionaries

What is a Dictionary?

- In data structure terms, a dictionary is better termed an associative array or associative list or a map.
- You can think if it as a list of pairs, where the first element of the pair, the key, is used to retrieve the second element, the value.
- Thus we map a key to a value.

Key-Value Pairs

- The key acts as a “lookup” to find the associated value.
- Just like a dictionary, you look up a word by its spelling to find the associated definition.
- A dictionary can be searched to locate the value associated with a key.

Python Dictionary

- Use the { } marker to create a dictionary
- Use the : marker to indicate key:value pairs:

```
contacts= {‘bill’: ‘353-1234’,
  ‘rich’: ‘269-1234’, ‘jane’: ‘352-
  1234’}
print contacts
{‘jane’: ‘352-1234’,
  ‘bill’: ‘353-1234’,
  ‘rich’: ‘369-1234’}
```

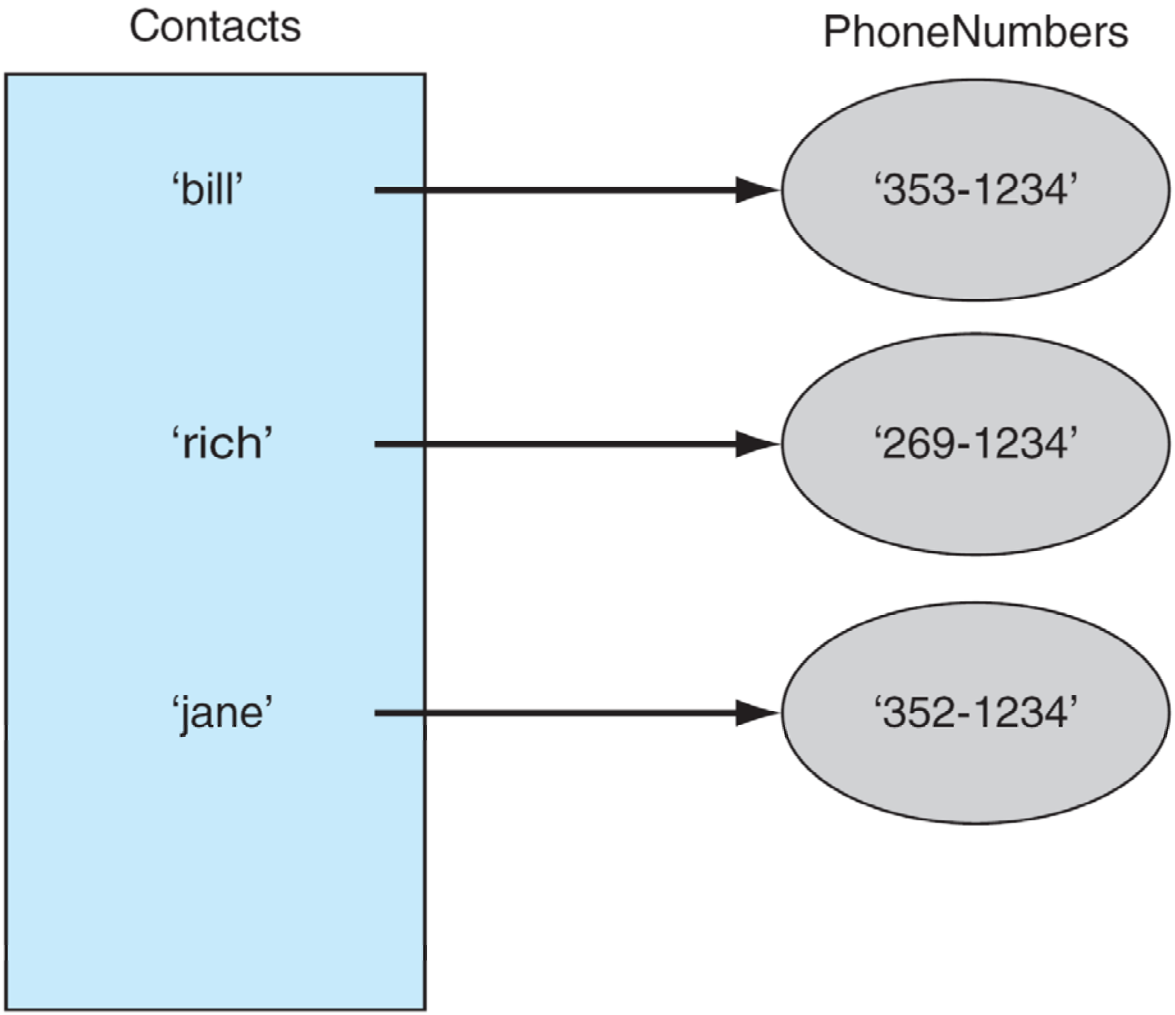


FIGURE 8.1 Phone contact list: names and phone numbers.

Keys and Values

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

Collections but not a Sequence

- Dictionaries are collections, but they are not sequences like 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!

```
myDict={}
```

– an empty dictionary

```
myDict[‘bill’]=25
```

– added the pair ‘bill’:25

```
print myDict[‘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

```
myDict = {‘bill’:3, ‘rich’:10}
print myDict[‘bill’]  # prints 3
myDict[‘bill’] = 100
print myDict[‘bill’]  # prints 100
```

Again, Common Operators

Like others, dictionaries respond to these:

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

Lots of Methods

- `myDict.items()` – all the key/value pairs
- `myDict.keys()` – all the keys
- `myDict.values()` – all the values
- `myDict.clear()` – empty the dictionary
- `myDict.copy()` – shallow copy

Dictionaries are Iterable

```
for key in myDict:
    print key
```

- prints all the keys

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

- prints all the key/value pairs

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

- prints all the values

Building Dictionaries

- Can build dictionaries from a list of tuples using the `dict` function:
  - `dict([(‘a’,1),(‘b’,2),(‘c’,3)])` yields
  - `{'a': 1, 'c': 3, 'b': 2}`

Building Dictionaries Faster

- `zip` creates pairs from two parallel lists:
  - `zip(“abc”,[1,2,3])` yields `[(‘a’,1),(‘b’,2),(‘c’,3)]`
- That’s good for building dictionaries. We call the `dict` function which takes a list of pairs to make a dictionary:
  - `dict(zip(“abc”,[1,2,3]))` yields
  - `{'a': 1, 'c': 3, 'b': 2}`

Sorting Dictionaries

- Remember the sorted() function?

```
>>> sorted(['a', 'b', 'd', 'c'])
['a', 'b', 'c', 'd']
```

- Sort by keys:
  - for key in sorted(myDict):

    print key, myDict[key]
- Sort by values:
  - for value in sorted(myDict.values()):

    print value

Example: Word Counts

- Prompt the user for input text, print each word and the number of occurrences of that word in the text.
- We can do this without dictionaries using lists and the string split(), find(), and/or replace() methods, but is this easier with dictionaries?

Example: Word Counts

- Create a dictionary with a count associated with each word.
- Iterate through the dictionary printing the words (keys) and counts (values).

Example: Most Common Word

- Prompt the user for input text, print the most common word in the text.

Example: Most Common Word

- Can use the max() function to find the largest count, but we need the key information.
- Loop through myDict.items(), keep track of key associated with largest value.
- Can also convert to a list of tuples and then call the list max() method (which uses the first element of tuples for comparison).

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

```
mySet = set(“abcd”)
```

- The “set” keyword creates a set.
- The single argument that follows must be *iterable*, that is, something that can be walked through one item at a time with a `for.`
- The result is a set data structure:

```
print mySet
set(['a', 'c', 'b', 'd'])
```

Diverse Elements

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

```
mySet = 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.

```
mySet = set(“aabbccdd”)
print mySet
set(['a', 'c', 'b', 'd'])
```

Common Operators

Most data structures respond to these:

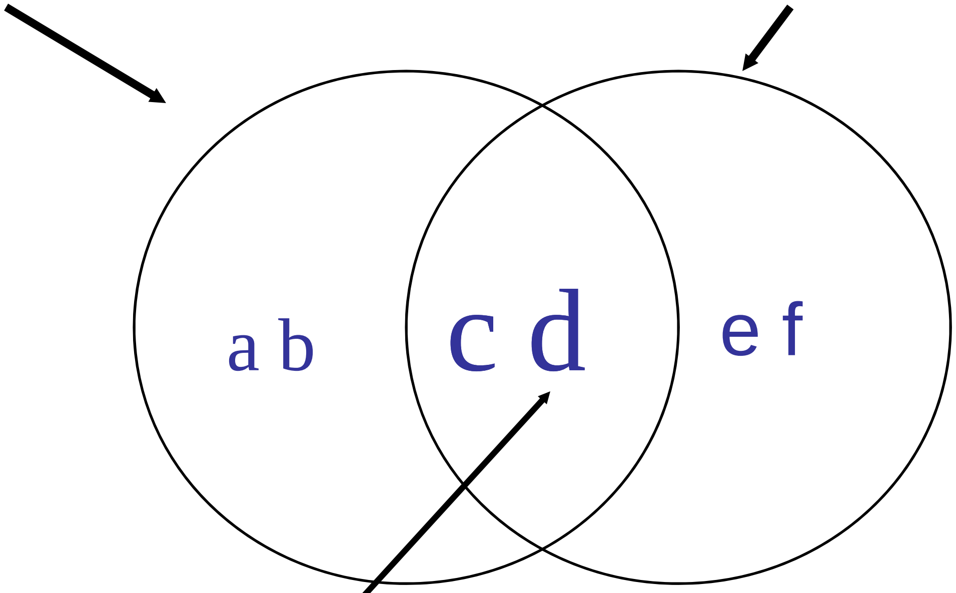
- `len(mySet)`
  - – the number of elements in a set
- `element in mySet`
  - – boolean indicating whether element is in the set
- `for element in mySet:`
  - – iterate through the elements in `mySet`

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.

Set Ops, Intersection

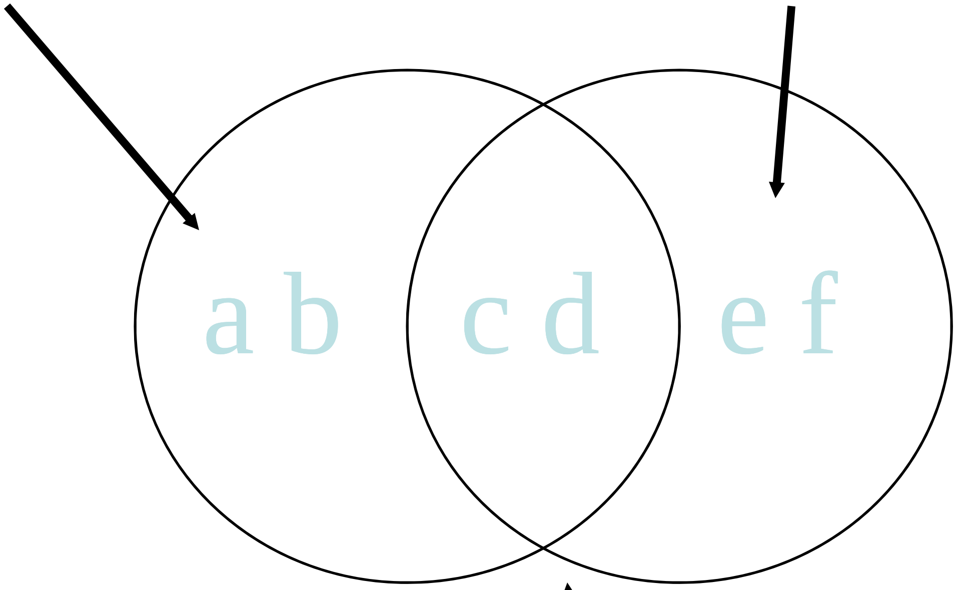
mySet=set(“abcd”); newSet=set(“cdef”)



mySet.intersection(newSet) returns set([‘c’,’d’])

Set Ops, Union

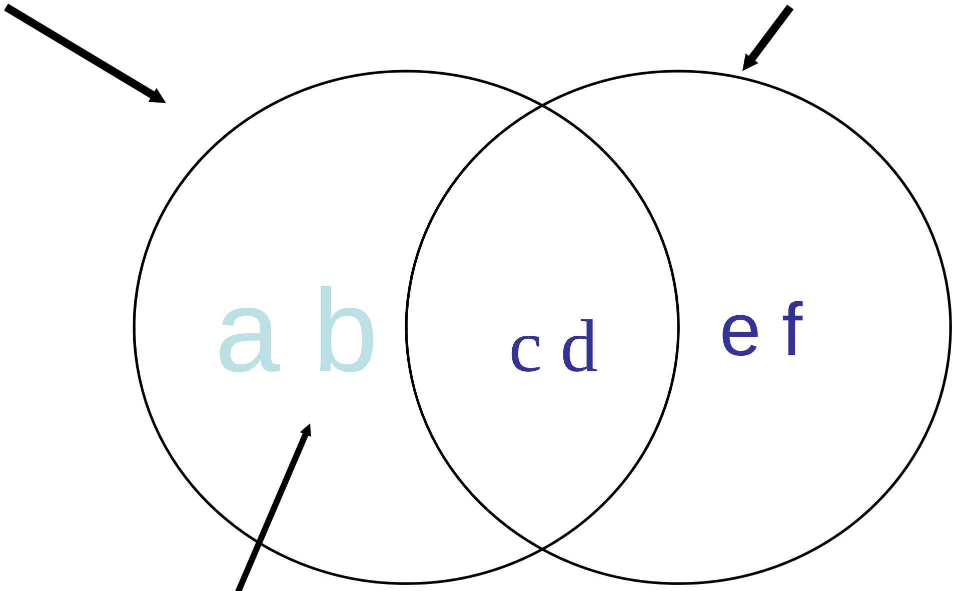
mySet=set(“abcd”); newSet=set(“cdef”)



mySet.union(newSet) returns set([‘a’,’b’,‘c’,’d’,’e’,’f’])

Set Ops, Difference

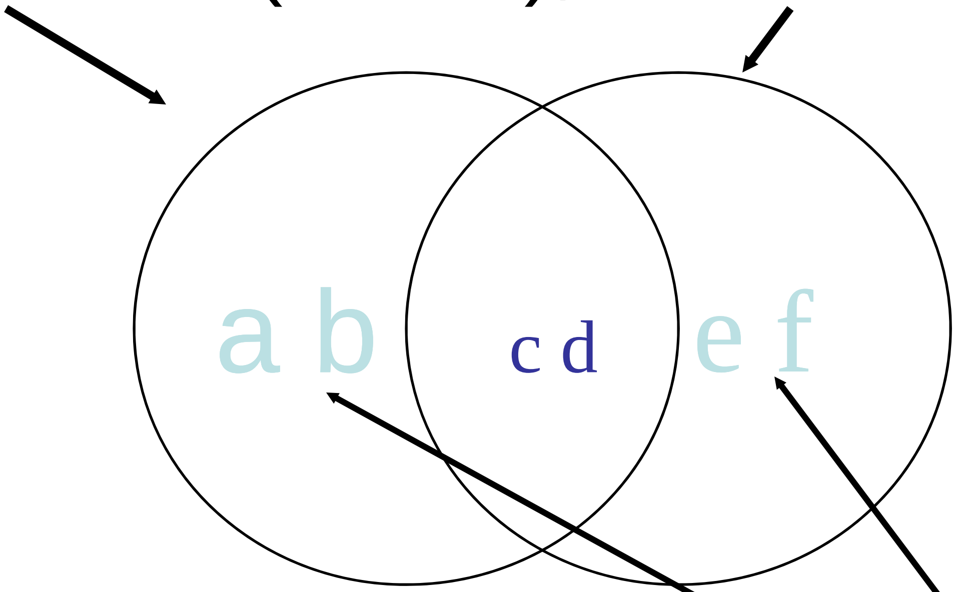
mySet=set(“abcd”); newSet=set(“cdef”)



`mySet.difference(newSet)` returns `set([‘a’,’b’])`

Set Ops, symmetric difference

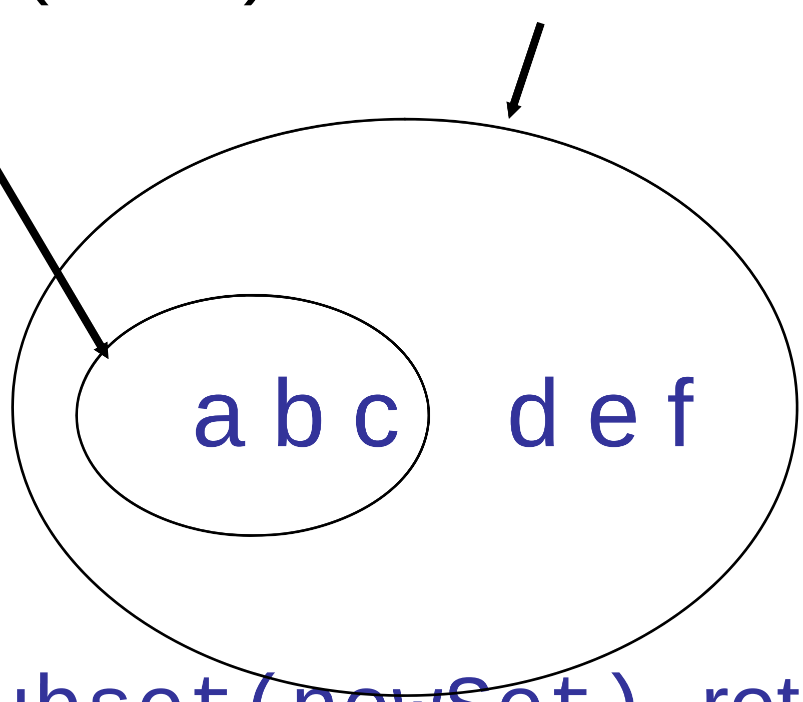
mySet=set(“abcd”); newSet=set(“cdef”)



```
mySet.symmetric_difference(newSet)
returns            set([‘a’,’b’,’e’,’f’])
```

Set Ops, super and sub sets

mySet=set(“abc”); newSet=set(“abcdef”)



mySet.issubset(newSet) returns True

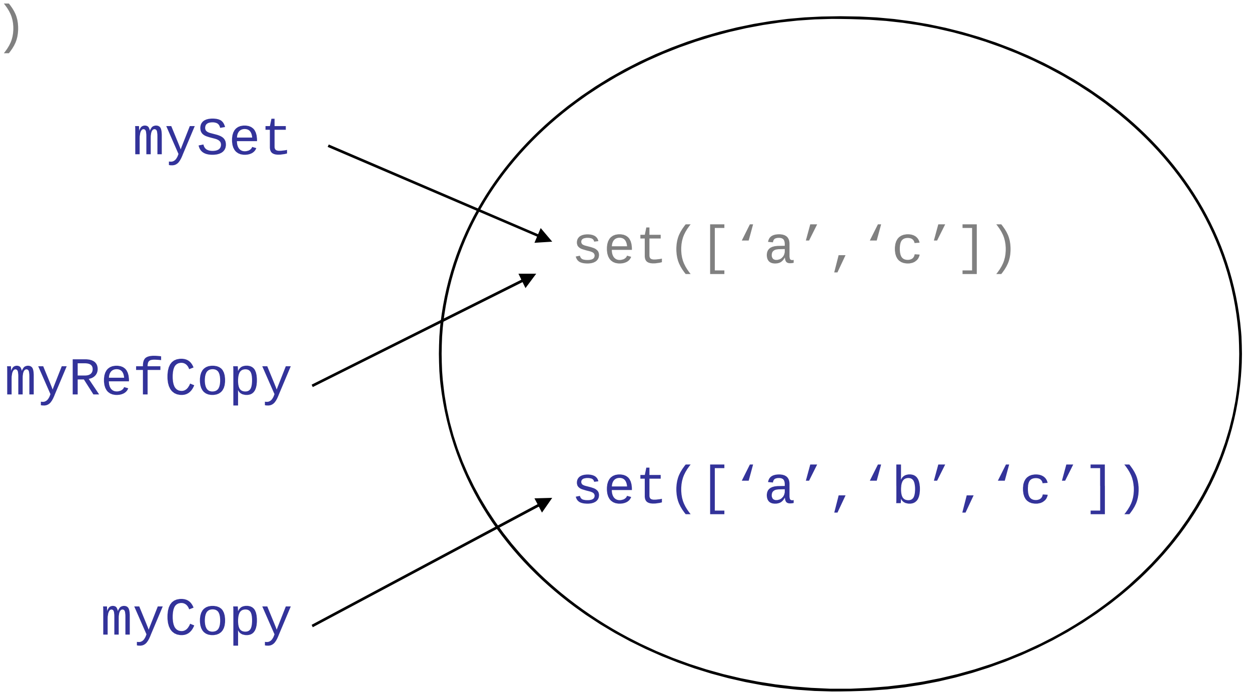
newSet.issuperset(mySet) returns True

Other Set Ops

- `mySet.add(“g”)`
  - Adds to the set, no effect if item is in set already.
- `mSet.clear()`
  - Empties the set.
- `mySet.remove(“g”)`
  - Removes “g” from the set.
- `mySet.copy()`
  - Returns a shallow copy of `mySet.`

Copy vs. Assignment

```
mySet=set(“abc”)
myCopy=mySet.copy()
myRefCopy=mySet
mySet.remove(‘b’)
```



mySet

myRefCopy

myCopy

set([‘a’,‘c’])

set([‘a’,‘b’,‘c’])

Example: Common Words

- Prompt user for two sentences, print words occurring in both sentences (print each word only once).
- We can certainly do this with dictionaries and/or lists.
- Is this easier with sets?