Dictionaries

More Data Structures

- We have seen the string and list data structures and their uses.
- In particular, the dictionary is an important, very useful part of Python as well as generally useful to solve many problems.

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'}
```

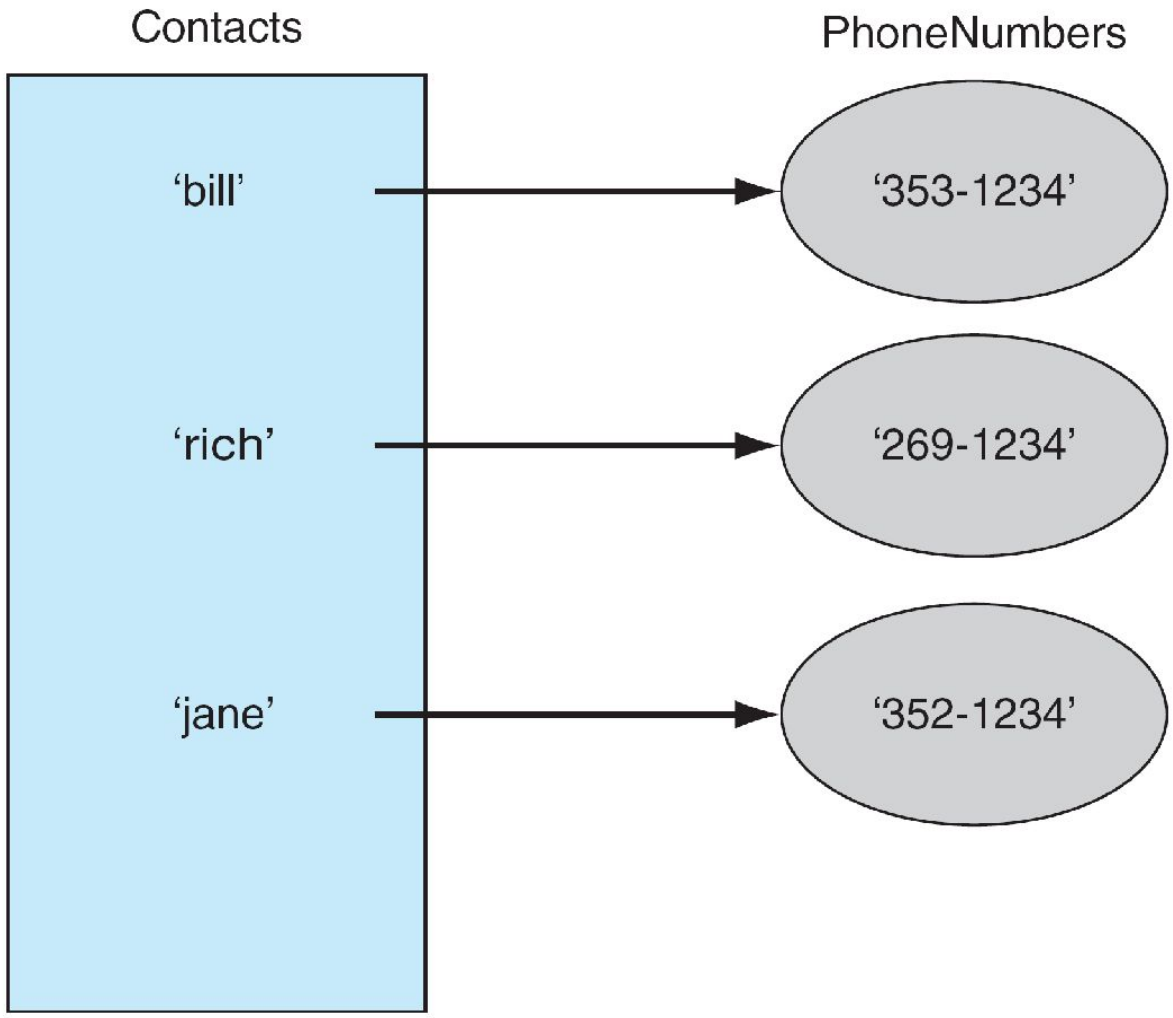

FIGURE 8.1 Phone contact list: names and phone numbers.

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
- `key in myDict`
  does the key exist in the dictionary
- `myDict.clear()` – empty the dictionary
- `myDict.update(yourDict)` – for each key in `yourDict`, updates `myDict` with that key/value pair

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

Doing something with this

- Write a function called letterCount that:
  - takes in a string as a parameter
  - prints a table of the letters of the alphabet (in alphabetical order) together with the number of times each letter occurs.
  - Case should be ignored.