Dictionaries in Python

• Dictionaries in Python are similar in syntax to lists. In both data models, the fundamental operation is selection, which is indicated using square brackets. The difference is that index values for a dictionary need not be integers.

• When you look up a value in a dictionary, you supply the key as a string expression using the square-bracket notation, as in

   map[key]

   If the key is defined in the dictionary, this selection returns the value. If no definition has been supplied, Python raises a KeyError exception.

• Dictionary selections are assignable. You can set the value associated with a key by executing an assignment statement:

   map[key] = value

The dictfile.py Module

```
def readDictionary(filename, separator=': '):
    """""
    Creates a dictionary by reading key-value pairs from the specified file in which the division between the key and value is marked by the specified separator, which defaults to a colon. The function discards any leading and trailing whitespace from both the key and the value.
    """
    dictionary = {}
    with open(filename) as f:
        for line in f:
            index = line.find(separator)
            key = line[:index].strip()
            value = line[index + len(separator):].strip()
            dictionary[key] = value
    return dictionary
```

Finding an Airport from its Code

```
# File: FindAirportCodes.py
from dictfile import readDictionary

def FindAirportCodes():
    airportDictionary = readDictionary('AirportCodes.txt')
    while True:
        code = input("Enter airport code: ")
        if code == ": break
        if code in airportDictionary:
            print(airportDictionary[code])
        else:
            print("There is no airport code " + code)

# Startup code
if __name__ == "__main__":
    FindAirportCodes()
```
Iterating Through Keys in an Object

- One of the common operations that clients need to perform when using a dictionary is to iterate through the keys.
- Python supports this operation using the `for` statement, which has the following form:
  ```python
  for key in dict:
      value = dict[key]
      ... code to work with the individual key and value ...
  ```
- You can also use the `items` method to iterate through the keys and values together:
  ```python
  for key, value in dict.items():
      ... code to work with the individual key and value ...
  ```

Finding Airports by Location

```python
# File: FindAirportsByLocation.py
from dictfile import readDictionary
def FindAirportsByLocation():
    airportDictionary = readDictionary('AirportCodes.txt')
    while True:
        str = input('Enter search string: ')
        if str == '': break
        for code, location in airportDictionary.items():
            print(code + ' : ' + location)
# Startup code
if __name__ == '__main__':
    FindAirportsByLocation()
```

Symbol Tables

- Programming languages make use of dictionaries in several contexts, of which one of the easiest to recognize is a symbol table, which keeps track of the correspondence between variable names and their values.
- The `SymbolTable.py` application in the text implements a simple test of a symbol table that reads lines from the console, each of which is one of the following commands:
  - A simple assignment statement of the form `var = number`.
  - A variable alone on a line, which displays the variable’s value.
- Before running the program, we’re going to add two new features:
  - The command `list`, which lists all the variables.
  - The command `quit`, which exits from the program.

Sample Run of `SymbolTable.py`

```
SymbolTable
+ pi = 3.14159
+ a = 2.71828
+ x = 2
+ pi
3.14159
+ a
2.71828
+ x
2
+ list
a = 2.71828
pi = 3.14159
x = 2
+ quit
```

The End