Dictionaries in Python

Jerry Cain
CS 106AX
October 28, 2024
slides leveraged from those constructed by Eric Roberts

Dictionaries in Python

- Like most modern programming languages, Python provides a data structure allowing us to associate pairs of data values. Although the more common term in computer science is ***map***, Python calls this structure a ***dictionary***.
- A dictionary associates a simple data value called a ***key*** (most often a string) with a ***value****,* which is often larger and more complex.
- Applications of the map idea exist everywhere in the real world. A classic example—which is where Python gets the name—is a dictionary. The keys are the words, and the values are the corresponding definitions.
- You'll be relying on strings, lists, and dictionaries for Assignment 5, which goes out on Wednesday.

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
  ```

Using Dictionaries in an Application

- Before going on to look at other applications, it seems worth going through the example from the text, which uses a dictionary to map three-letter airport codes to their locations.
- The association list is stored in a text file that looks like this:

```
ATL: Atlanta, GA, USA
PEK: Beijing, China
DXB: Dubai, United Arab Emirates
LAX: Los Angeles, CA, USA
HND: Tokyo, Japan
ORD: Chicago, IL, USA
LHR: London, England, United Kingdom
HKG: Hong Kong, Hong Kong
         :
```

- The **`dictfile.py`** module shows how to read this type of data file into a Python object.

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:

```
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:

```
for key, value in dict.items():
    . . . code to work with the individual key and value . . .
```

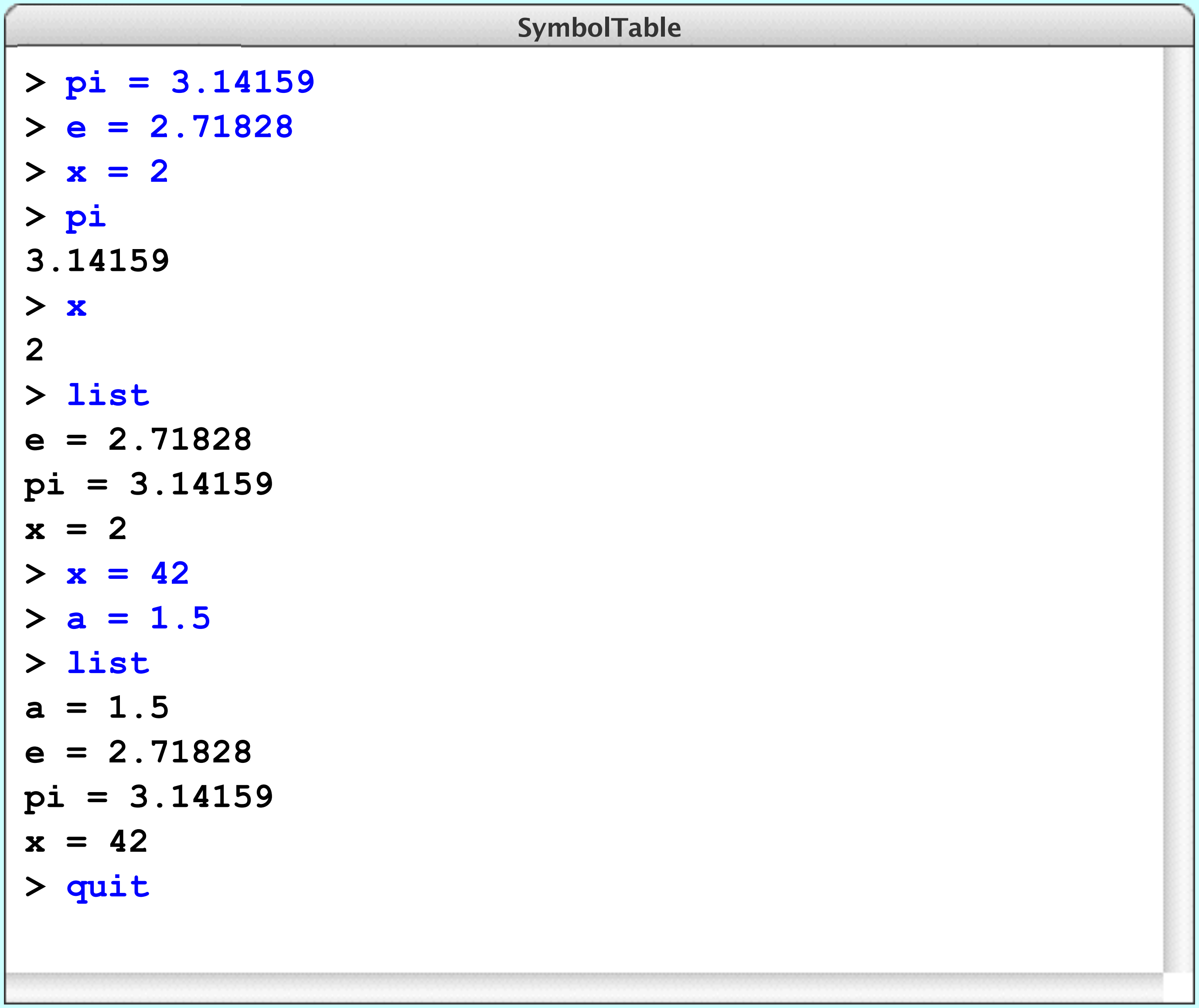
Finding Airports by Location

```
# 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():
            if str in location:
                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`



```
> pi = 3.14159
> e = 2.71828
> x = 2
> pi
3.14159
> x
2
> list
e = 2.71828
pi = 3.14159
x = 2
> x = 42
> a = 1.5
> list
a = 1.5
e = 2.71828
pi = 3.14159
x = 42
> quit
```

The End