Programming with Python

Lists, Tuples and Dictionaries

Lists and Dictionaries

- A **list** allows the programmer to manipulate a sequence of data values of any types
- A **dictionary** organizes data values by association with other data values rather than by sequential position
- Lists and dictionaries provide powerful ways to organize data in useful and interesting applications

Lists

- List: Sequence of data values (**items** or **elements**)
- Some examples:
  - Shopping list for the grocery store
  - Guest list for a wedding
  - Recipe, which is a list of instructions
  - Text document, which is a list of lines
- Each item in a list has a unique **index** that specifies its position (from 0 to length – 1)

List Literals and Basic Operators

- Some examples:

```
['apples', 'oranges', 'cherries']

[[5, 9], [541, 78]]
```

- When an element is an expression, its value is included in the list:

```
>>> x = 2
>>> [x, math.sqrt(x)]
[2, 1.4142135623730951]
```

Lists of integers can be built using **`range()`** **function**:

```
>>> first = [1, 2, 3, 4]
>>> second = list(range(1, 5))
>>> first
[1, 2, 3, 4]
>>> second
[1, 2, 3, 4]
>>>
```

List Literals and Basic Operators

- Some examples:

```
['apples', 'oranges', 'cherries']

[[5, 9], [541, 78]]
```

- When an element is an expression, its value is included in the list:

```
>>> x = 2
>>> [x, math.sqrt(x)]
[2, 1.4142135623730951]
```

Lists of integers can be built using **`range()`** function:

```
>>> first = [1, 2, 3, 4]
>>> second = list(range(1, 5))
>>> first
[1, 2, 3, 4]
>>> second
[1, 2, 3, 4]
>>>
```

List Literals and Basic Operators

- **`len`**, **`[]`**, **`+`**, and **==** work on lists as expected:

```
>>> len(first)
4
>>> first[2:4]
[3, 4]
>>> first + [5, 6]
[1, 2, 3, 4, 5, 6]
>>> first == second
True
```

- To print the contents of a list:

```
>>> print("1234")
1234
>>> print([1, 2, 3, 4])
[1, 2, 3, 4]
>>>
```

- **`in`** detects the presence of an element:

```
>>> 0 in [1, 2, 3]
False
```

Replacing an Element in a List

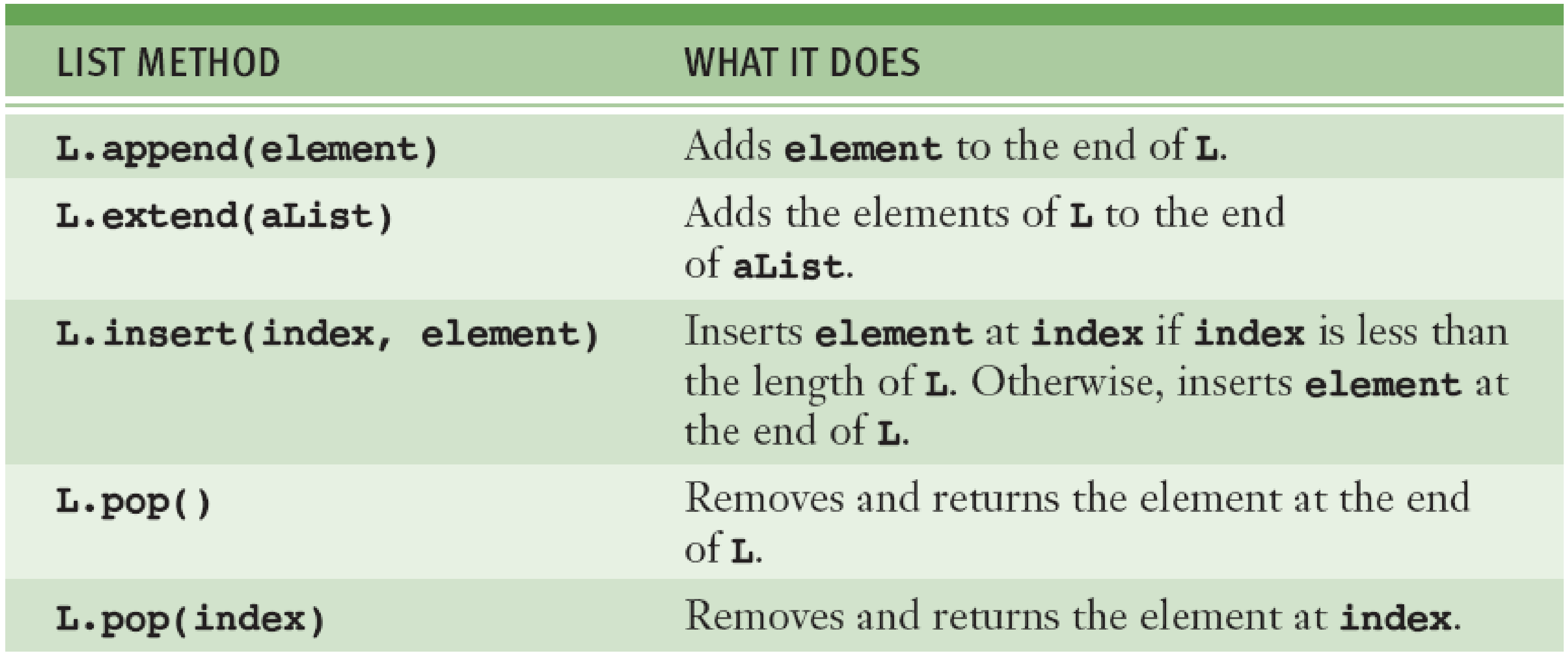
- A list is **mutable**
  - Elements can be inserted, removed, or replaced
  - The list itself maintains its identity, but its **state**—its length and its contents—can change
- Subscript operator is used to replace an element:

```
>>> example = [1, 2, 3, 4]
>>> example
[1, 2, 3, 4]
>>> example[3] = 0
>>> example
[1, 2, 3, 0]
```

- Subscript is used to reference the **target** of the assignment, which is not the list but an element's position within it

List Methods for Inserting and Removing Elements

- The **`list`** type includes several methods for inserting and removing elements



LIST METHOD	WHAT IT DOES
`L.append(element)`	Adds **`element`** to the end of **`L`**.
`L.extend(aList)`	Adds the elements of **`L`** to the end of **`aList`**.
`L.insert(index, element)`	Inserts **`element`** at **`index`** if **`index`** is less than the length of **`L`**. Otherwise, inserts **`element`** at the end of **`L`**.
`L.pop()`	Removes and returns the element at the end of **`L`**.
`L.pop(index)`	Removes and returns the element at **`index`**.

[TABLE 5.2] List methods for inserting and removing elements

Searching a List

- `in` determines an element's presence or absence, but does not return position of element (if found)
- Use method `index` to locate an element's position in a list
  - Raises an error when the target element is not found

```
aList = [34, 45, 67]
target = 45
if target in aList:
    print(aList.index(target))
else:
    print(-1)
```

Sorting a List

- A list's elements are always ordered by position, but you can impose a **natural ordering** on them
  - For example, in alphabetical order
- When the elements can be related by comparing them <, >, and ==, they can be sorted
  - The method **`sort`** mutates a list by arranging its elements in ascending order

```
>>> example = [4, 2, 10, 8]
>>> example
[4, 2, 10, 8]
>>> example.sort()
>>> example
[2, 4, 8, 10]
```

Tuples

- A **tuple** resembles a list, but is immutable
  - Indicate by enclosing its elements in `()`

- Most of the operators and functions used with lists can be used in a similar fashion with tuples

- What is the advantage of tuple over list?

Dictionaries

- A dictionary organizes information by **association**, not position
  - Example: When you use a dictionary to look up the definition of “mammal,” you don’t start at page 1; instead, you turn to the words beginning with “M”
- Data structures organized by association are also called **tables** or **association lists**
- In Python, a **dictionary** associates a set of **keys** with data values

Dictionary Literals

- A Python dictionary is written as a sequence of key/value pairs separated by commas
  - Pairs are sometimes called **entries**
  - Enclosed in curly braces (**{** and **}**)
  - A colon (**:**) separates a key and its value
- Examples:

```
{'Sarah':'476-3321', 'Nathan':'351-7743'}

{'Name':'Molly', 'Age':18}

{}
```

- Keys can be data of any immutable types, including other data structures

Adding Keys and Replacing Values

- Add a new key/value pair to a dictionary using **`[]`**:

```
<a dictionary>[<a key>] = <a value>
```

- Example:

```
>>> info = {}
>>> info["name"] = "Sandy"
>>> info["occupation"] = "hacker"
>>> info
{'name': 'Sandy', 'occupation': 'hacker'}
>>>
```

- Use **`[]`** also to replace a value at an existing key:

```
>>> info["occupation"] = "manager"
>>> info
{'name': 'Sandy', 'occupation': 'manager'}
>>>
```

Accessing Values

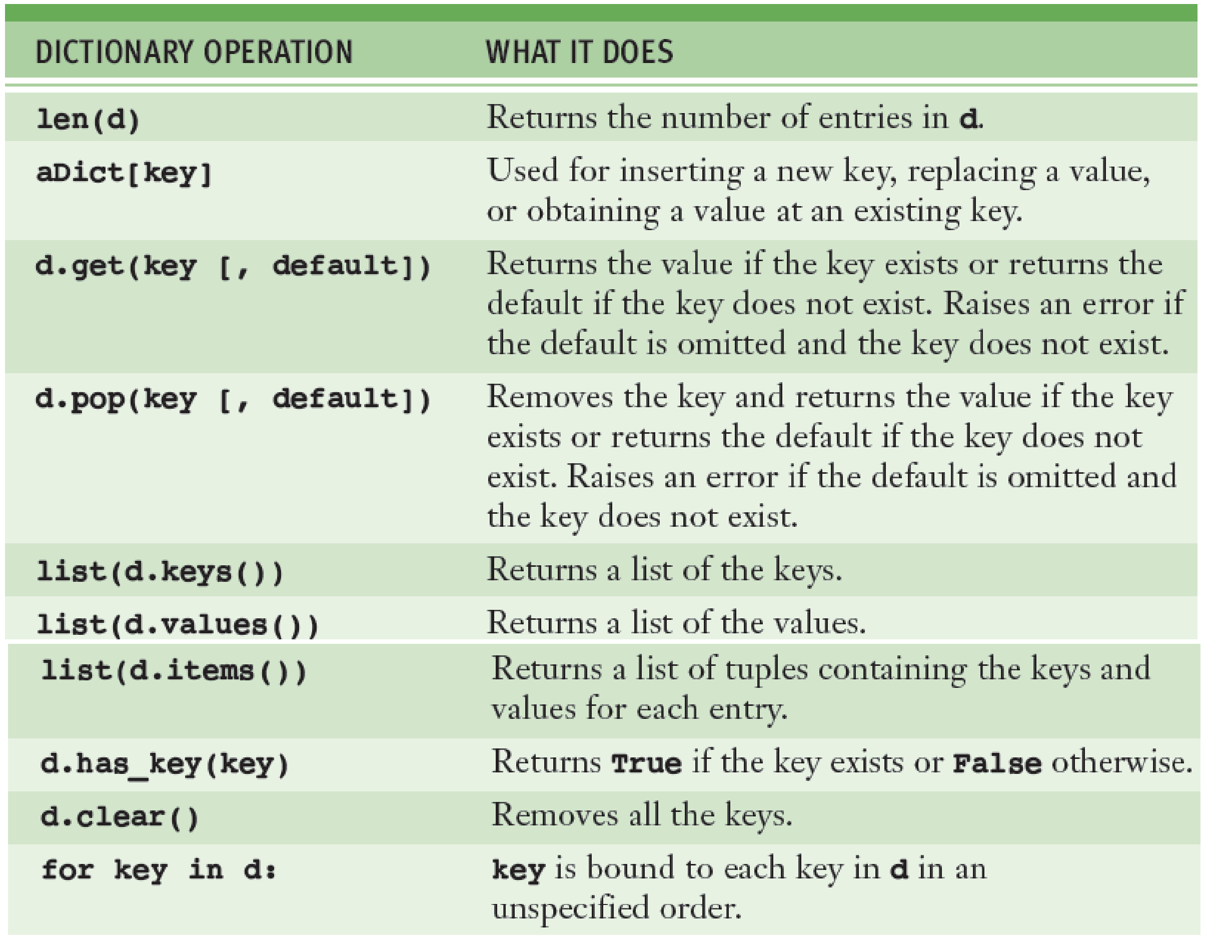
- Use **`[]`** to obtain the value associated with a key
  - If key is not present in dictionary, an error is raised

```
>>> info["name"]
'Sandy'
>>> info["job"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'job'
>>>
```

- If the existence of a key is uncertain, test for it using the dictionary method **`has_key`**
  - Easier strategy is to use the method **`get`**

```
>>> print(info.get("job", None))
None
>>>
```

Traversing a Dictionary

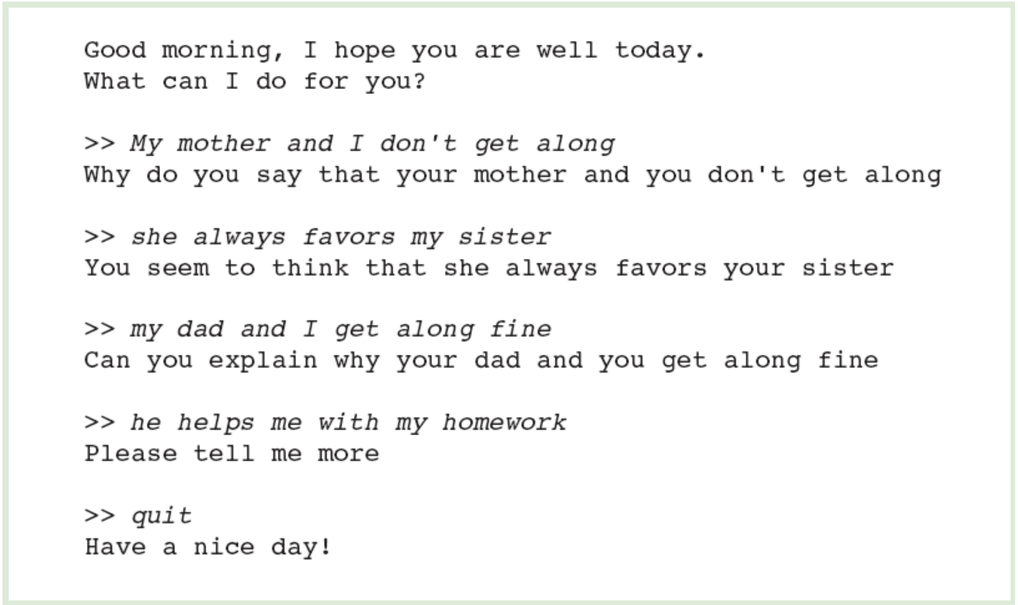


DICTIONARY OPERATION	WHAT IT DOES
`len(d)`	Returns the number of entries in **`d`**.
`aDict[key]`	Used for inserting a new key, replacing a value, or obtaining a value at an existing key.
`d.get(key [, default])`	Returns the value if the key exists or returns the default if the key does not exist. Raises an error if the default is omitted and the key does not exist.
`d.pop(key [, default])`	Removes the key and returns the value if the key exists or returns the default if the key does not exist. Raises an error if the default is omitted and the key does not exist.
`list(d.keys())`	Returns a list of the keys.
`list(d.values())`	Returns a list of the values.
`list(d.items())`	Returns a list of tuples containing the keys and values for each entry.
`d.has_key(key)`	Returns **`True`** if the key exists or **`False`** otherwise.
`d.clear()`	Removes all the keys.
`for key in d:`	**`key`** is bound to each key in **`d`** in an unspecified order.

[TABLE 5.4] Some commonly used dictionary operations

Case Study: Nondirective Psychotherapy

- Doctor in this kind of therapy responds to patient's statements by rephrasing them or indirectly asking for more information
- Request:
  - Write a program that emulates a nondirective psychotherapist



```
Good morning, I hope you are well today.
What can I do for you?

>> My mother and I don't get along
Why do you say that your mother and you don't get along

>> she always favors my sister
You seem to think that she always favors your sister

>> my dad and I get along fine
Can you explain why your dad and you get along fine

>> he helps me with my homework
Please tell me more

>> quit
Have a nice day!
```

[FIGURE 5.4] A session with the doctor program

Case Study: Nondirective Psychotherapy (Analysis)

- When user enters a statement, program responds in one of two ways:
  - With a randomly chosen hedge, such as “Please tell me more”
  - By changing some key words in user’s input string and appending string to a randomly chosen qualifier
    - Thus, to “My teacher always plays favorites,” program might reply, “Why do you say that your teacher always plays favorites?”

Case Study: Nondirective Psychotherapy (Design)

- Program consists of a set of collaborating functions that share a common data pool
- Pseudocode:

```
output a greeting to the patient
while True
    prompt for and input a string from the patient
    if the string equals "Quit"
        output a sign-off message to the patient
        break
    call another function to obtain a reply to this string
    output the reply to the patient
```

Case Study: Implementation

```python
"""
Program: doctor.py
Author: Ken
Conducts an interactive session of nondirective psychotherapy.
"""
import random

hedges = ("Please tell me more.",
          "Many of my patients tell me the same thing.",
          "Please continue.")

qualifiers = ("Why do you say that ",
              "You seem to think that ",
              "Can you explain why ")

replacements = {"I":"you", "me":"you", "my":"your",
                "we":"you", "us":"you", "mine":"yours"}
```

Case Study: Implementation

```python
def reply(sentence):
    """Builds and returns a reply to the sentence."""
    probability = random.randint(1, 4)
    if probability == 1:
        return random.choice(hedges)
    else:
        return random.choice(qualifiers) + changePerson(sentence)

def changePerson(sentence):
    """Replaces first person pronouns with second person
    pronouns."""
    words = sentence.split()
    replyWords = []
    for word in words:
        replyWords.append(replacements.get(word, word))
    return " ".join(replyWords)

def main():
    """Handles the interaction between patient and doctor."""
    print("Good morning, I hope you are well today.")
    print("What can I do for you?")
    while True:
        sentence = input("\n>> ")
        if sentence.upper() == "QUIT":
            print("Have a nice day!")
            break
        print(reply(sentence))

main()
```

Homework

- Complete and test the Nondirective Psychotherapy program if you have not finished it in class.
- Enhance the program by introducing a tuple of exclamations and integrate the exclamations into doctor's replies:

```
exclamations = (“Oh, no!”, “Really?”, “Unbelievable!”)
```