Chapter 10

- **Dictionaries and Data Sets**

Topics

- Dictionaries
- Sets
- Serializing Objects

Dictionaries

- **<u>Dictionary</u>: object that stores a collection of data**
  - Each element consists of a *key* and a *value*
    - Often referred to as *mapping* of key to value
    - Key must be an immutable object
  - To retrieve a specific value, use the key associated with it
  - Format for creating a dictionary

```
dictionary =
        {key1:val1, key2:val2}
```

Dictionaries

CONCEPT: A dictionary is an object that stores a collection of data.

- Each element in a dictionary has two parts: a key and a value. You use a key to locate a specific value.
- This is similar to the process of looking up a word in the Merriam-Webster dictionary, where the words are keys and the definitions are values.
- We could create a dictionary in which each element contains an employee ID number as the key and that employee's name as the value.
- Another example would be a program that lets us enter a person's name and gives us that person's phone number. A person's name as the key and that person's phone number as the value

<u>Creating a Dictionary</u>

phonebook = {' Chris' :' 555-1111' , ' Katie' :' 555-2222' , ' Joanne' :' 555-3333' }

(the first element is Chris' :' 555-1111'. the key is ' Chris' and the value is ' 555-1111')

In this example the keys and the values are strings. The values in a dictionary can be objects of any type,

Retrieving a Value from a Dictionary

- **Elements in dictionary are unsorted**
- **General format for retrieving value from dictionary: `dictionary[key]`**
  - If `key` in the dictionary, associated value is returned, otherwise, `KeyError` exception is raised
- **Test whether a key is in a dictionary using the `in` and `not in` operators**
  - Helps prevent `KeyError` exceptions

Retrieving a Value from a Dictionary

The elements in a dictionary are not stored in any particular order. For example, look at the following interactive session in which a dictionary is created and its elements are displayed:

```
>>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
>>> phonebook
{' Chris' : ' 555-1111' , ' Joanne' : ' 555-3333' , ' Katie' : ' 555-2222' }
```

To <u>retrieve a value</u> from a dictionary, you simply write an expression in the following general format:

dictionary_name[key]

```
1 >>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
2 >>> phonebook[' Chris' ]
3 ' 555-1111‘
```

If the key exists in the dictionary, the expression returns the value that is associated with the key<u>. If the key does not exist</u>, a KeyError exception is raised.

```
>>> phonebook[' Kathryn' ]
Traceback (most recent call last): File "<pyshell#5>", line 1, in <module> phonebook[' Kathryn' ] KeyError: ' Kathryn‘
```

NOTE: <u>string comparisons</u> are <u>case sensitive</u>

Adding Elements to an Existing Dictionary

- **Dictionaries are mutable objects**
- **To add a new key-value pair:**

```
dictionary[key] = value
```

  – If key exists in the dictionary, the value associated with it will be changed

Adding Elements to an Existing Dictionary

dictionary_name[key] = value

In the general format, **dictionary_name** is the variable that references the dictionary, and **key** is a key. If key already exists in the dictionary, its associated value will be changed to value. If the key does not exist, it will be added to the dictionary, along with value as its associated value.

<u>**Example**</u>:

```
1 >>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
2 >>> phonebook[' Joe' ] = ' 555-0123'
3 >>> phonebook[' Chris' ] = ' 555-4444'
4 >>> phonebook
5 {'Chris': '555-4444', 'Joanne': '555-3333', 'Joe': '555-0123', 'Katie': '555-2222'}
```

Line 2 adds a new key-value pair to the phonebook dictionary. Because there is no key ' Joe' in the dictionary, this statement adds the key ' Joe' , along with its associated value ' 555-0123' .

Line 3 changes the value that is associated with an existing key. Because the key ' Chris' already exists in the phonebook dictionary, this statement changes its associated value to ' 555-4444' .

Deleting Elements From an Existing Dictionary

- **To delete a key-value pair:**

  `del` ***`dictionary`*****`[`*****`key`*****`]`**

  – If key is not in the dictionary, `KeyError` exception is raised

Deleting Elements From an Existing Dictionary

del dictionary_name[key]

In the general format, dictionary_name is the variable that references the dictionary, and key is a key. After the statement executes, the key and its associated value will be deleted from the dictionary. If the key does not exist, a KeyError exception is raised

```
1 >>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'} e
2 >>> phonebook e
3 {' Chris' : ' 555-1111' , ' Joanne' : ' 555-3333' , ' Katie' : ' 555-2222' }
4 >>> del phonebook[' Chris' ]
5 >>> phonebook
6 {' Joanne' : ' 555-3333' , ' Katie' : ' 555-2222' }
7 >>> del phonebook[' Chris' ] e
8 Traceback (most recent call last):   File "<pyshell#5>", line 1, in <module> del phonebook[' Chris' ]
KeyError: ' Chris‘
```

Line 4 deletes the element with the key ' Chris' , and line 5 displays the contents of the dictionary. You can see in the output in line 6 that the element no longer exists in the dictionary.

Line 7 tries to delete the element with the key ' Chris' again. Because the element no longer exists, a KeyError exception is raised.

Deleting Elements From an Existing Dictionary (cont.)

To prevent a KeyError exception from being raised, you should use the in operator to determine whether a key exists before you try to delete it and its associated value.

```
>>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
>>> if ' Chris'  in phonebook:
            del phonebook[' Chris' ]

>>> phonebook
{' Joanne' : ' 555-3333' , ' Katie' : ' 555-2222' }
```

Getting the Number of Elements and Mixing Data Types

- `len` function: used to obtain number of elements in a dictionary
- **Keys must be immutable objects, but associated values can be any type of object**
  - One dictionary can include keys of several different immutable types
- **Values stored in a single dictionary can be of different types**

Getting the Number of Elements and Mixing Data Types

```
1   >>> phonebook = {' Chris' :' 555-1111' , ' Katie' :' 555-2222' } e
2   >>> num_items = len(phonebook)
3   >>> print(num_items)
4   2
```

Line 1 creates a dictionary with two elements and assigns it to the phonebook variable.

Line 2 calls the len function passing the phonebook variable as an argument. The function returns the value 2, which is assigned to the num_items variable**.**

Line 3 passes num_items to the print function. The function's output is shown in line 4.

Creating an Empty Dictionary and Using `for` Loop to Iterate Over a Dictionary

- **To create an empty dictionary:**
  - Use `{ }`
  - Use built-in function **`dict()`**
  - Elements can be added to the dictionary as program executes
- **Use a `for` loop to iterate over a dictionary**
  - General format: `for` *`key`* `in` *`dictionary`*`:`

Creating an Empty Dictionary

Sometimes you need to create an empty dictionary and then add elements to it as the program executes. You can use an empty set of curly braces to create an empty dictionary

```
1 >>> phonebook = {}
2 >>> phonebook[' Chris' ] = ' 555-1111'
3 >>> phonebook[' Katie' ] = ' 555-2222'
4 >>> phonebook[' Joanne' ] = ' 555-3333'
5 >>> phonebook
6 {' Chris' : ' 555-1111' , ' Joanne' : ' 555-3333' , ' Katie' : ' 555-
2222' }
7 >>>
```

The statement in line 1 creates an empty dictionary and assigns it to the phonebook variable. Lines 2 through 4 add key-value pairs to the dictionary, and the statement in line 5 displays the dictionary's contents.

You can also use the built-in dict() method to create an empty dictionary, as shown in the following statement:

```
phonebook = dict()
```

After this statement executes, the phonebook variable will reference an empty dictionary.

Using `for` Loop to Iterate Over a Dictionary

You can use the for loop to iterate over all the keys in a dictionary

This loop iterates once for each element in the dictionary. Each time the loop iterates, the variable **key** is assigned a key.

Example:

```
>>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
>>> for key in phonebook:
        print(key)
Chris
Joanne
Katie
```

Another **for loop** that iterates once for each element of the phonebook dictionary, assigning a key to the key variable

```
>>> for key in phonebook:
        print(key, phonebook[key])
Chris 555-1111
Joanne 555-3333
Katie 555-2222
```

Some Dictionary Methods

- **<u>`clear` method</u>: deletes all the elements in a dictionary, leaving it empty**
  - Format: `*dictionary*.clear()`
- **<u>`get` method</u>: gets a value associated with specified key from the dictionary**
  - Format: `*dictionary*.get(*key*, *default*)`
    - *`default`* is returned if *`key`* is not found
  - Alternative to `[]` operator
    - Cannot raise `KeyError` exception

Some Dictionary Methods (cont'd.)

- **`items` method: returns all the dictionaries keys and associated values**
  - Format: `dictionary.items()`
  - Returned as a *dictionary view*
    - Each element in dictionary view is a tuple which contains a key and its associated value
    - Use a `for` loop to iterate over the tuples in the sequence
      - Can use a variable which receives a tuple, or can use two variables which receive key and value

Some Dictionary Methods (cont'd.)

- **`keys` method: returns all the dictionaries keys as a sequence**
  - Format: `*dictionary*.keys()`
- **`pop` method: returns value associated with specified key and removes that key-value pair from the dictionary**
  - Format: `*dictionary*.pop(*key*, *default*)`
    - *`default`* is returned if *`key`* is not found

Some Dictionary Methods (cont'd.)

- `popitem` **method: returns a randomly selected key-value pair and removes that key-value pair from the dictionary**
  - Format: *`dictionary`*`.popitem()`
  - Key-value pair returned as a tuple
- `values` **method: returns all the dictionaries values as a sequence**
  - Format: *`dictionary`*`.values()`
  - Use a `for` loop to iterate over the values

Some Dictionary Methods (Clear method)

dictionary.clear()

```
>>> phonebook = {' Chris' :' 555-1111' , ' Katie' :' 555-2222' }
>>> phonebook
{' Chris' : ' 555-1111' , ' Katie' : ' 555-2222' }
>>> phonebook.clear()
>>> phonebook
{}
```

Some Dictionary Methods (Get method)

The get Method

You can use the get method as an alternative to the [] operator for getting a value from a dictionary. The get method does not raise an exception if the specified key is not found.

dictionary.get(key, default)

In the general format, dictionary is the name of a dictionary, key is a key to search for in the dictionary, and default is a default value to return if the key is not found. When the method is called, it returns the value that is associated with the specified key. If the specified key is not found in the dictionary, the method returns default.

```
>>> phonebook = {' Chris' :' 555-1111' , ' Katie' :' 555-2222' }
>>> value = phonebook.get(' Katie' , ' Entry not found' )
>>> print(value)
555-2222
>>> value = phonebook.get(' Andy' , ' Entry not found' )
>>> print(value)
Entry not found
>>>
```

Some Dictionary Methods (Items method)

The items Method

The items method returns all of a dictionary’s keys and their associated values. They are returned as a special type of sequence known as a dictionary view. Each element in the dictionary view is a tuple, and each tuple contains a key and its associated value. For example, suppose we have created the following dictionary:

phonebook = {'Chris':'555-1111', 'Katie':'555-2222' , 'Joanne':'555-3333'}

If we call the phonebook.items() method, it returns the following sequence:

[(' Chris' , ' 555-1111'), (' Joanne' , ' 555-3333'), (' Katie' , ' 555-2222')]

You can use the for loop to iterate over the tuples in the sequence

```
>>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222' , 'Joanne':'555-3333'}
>>> for key, value in phonebook.items():
                print(key, value)

Chris 555-1111
Joanne 555-3333
Katie 555-2222
```

Some Dictionary Methods (Keys method)

The **keys method** returns all of a dictionary's keys as a dictionary view, which is a type of sequence. Each element in the dictionary view is a key from the dictionary.

For example, suppose we have created the following dictionary:

phonebook = {' Chris' :' 555-1111' , ' Katie' :' 555-2222' , 'Joanne' :' 555-3333' }

If we call the **phonebook.keys()** method, it will return the following sequence:

[' Chris' , ' Joanne' , ' Katie']

The following interactive session shows how you can use a for loop to iterate over the sequence that is returned from the keys method:

```
>>> phonebook = {' Chris' :' 555-1111' ,
                ' Katie' :' 555-2222' ,
                ' Joanne' :' 555-3333' }
>>> for key in phonebook.keys():
                print(key)


Chris
Joanne
Katie
>>
```

Addison-Wesley is an imprint of PEARSON

Some Dictionary Methods (Pop method)

The **pop method** returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.

Here is the method’s general format:

dictionary.pop(key, default)

In the general format, dictionary is the name of a dictionary, key is a key to search for in the dictionary, and default is a default value to return if the key is not found.

When the method is called, it returns the value that is associated with the specified key, and it removes that key-value pair from the dictionary. If the specified key is not found in the dictionary, the method returns default.

```
>>> phonebook = {' Chris' :' 555-1111' , ' Katie' :' 555-2222' ,' Joanne' :' 555-3333'}
>>> phone_num = phonebook.pop(' Chris' , ' Entry not found' )
>>> phone_num = phonebook.pop(' Chris' , ' Entry not found' )
>>> phone_num
' 555-1111'
>>> phonebook
{' Joanne' : ' 555-3333' , ' Katie' : ' 555-2222' }
>>> phone_num = phonebook.pop(' Andy' , ' Element not found' )
>>> phone_num
' Element not found'
>>> phonebook
{' Joanne' : ' 555-3333' , ' Katie' : ' 555-2222' }
```

Some Dictionary Methods (cont'd.)

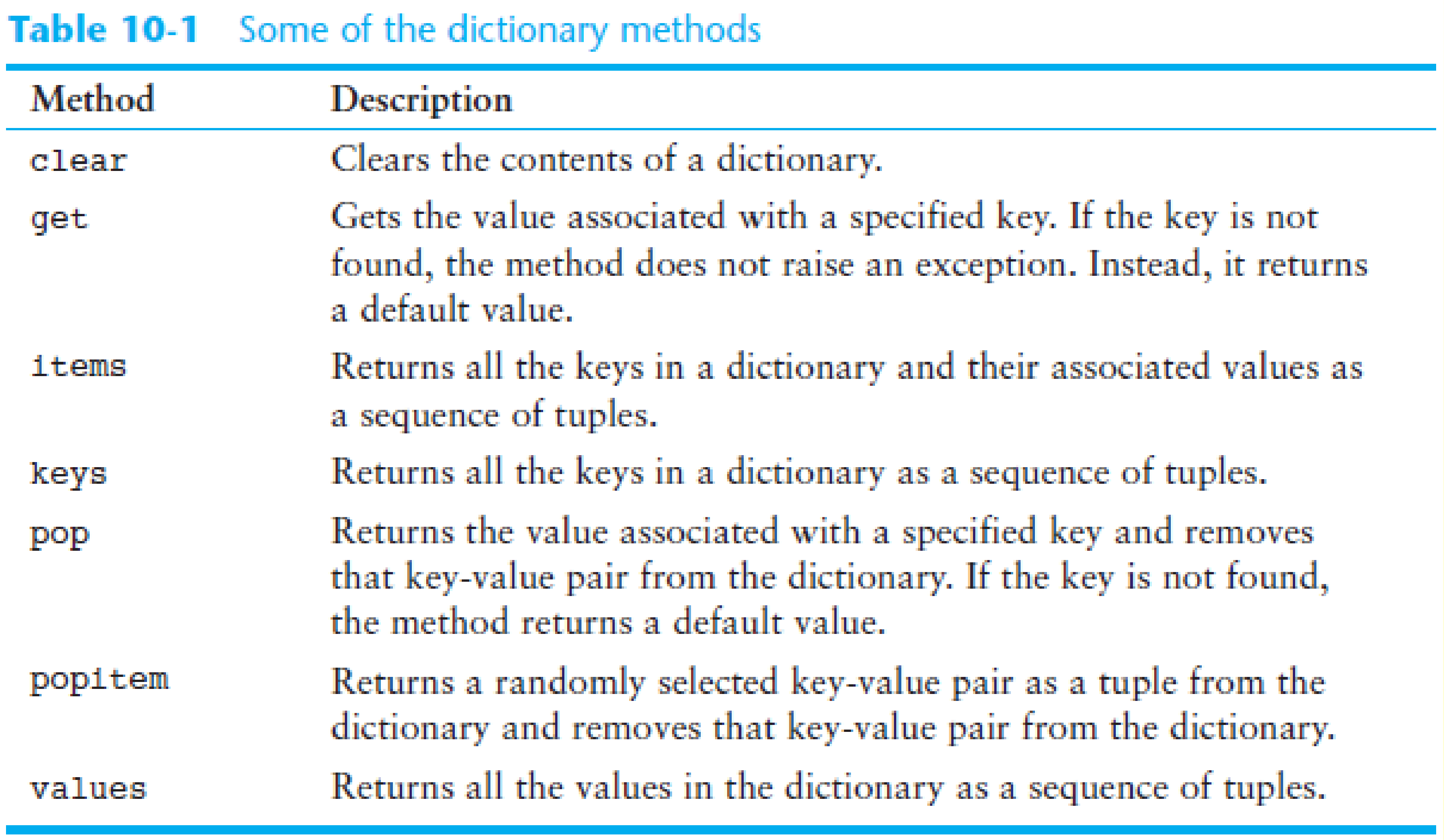

Table 10-1 Some of the dictionary methods

Method	Description
`clear`	Clears the contents of a dictionary.
`get`	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
`items`	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
`keys`	Returns all the keys in a dictionary as a sequence of tuples.
`pop`	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
`popitem`	Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.
`values`	Returns all the values in the dictionary as a sequence of tuples.

Sets

- **<u>Set</u>: object that stores a collection of data in same way as mathematical set**
  - All items must be unique
  - Set is unordered
  - Elements can be of different data types

Creating a Set

- **<u>`set` function</u>: used to create a set**
  - For empty set, call `set()`
  - For non-empty set, call `set(`*`argument`*`)` where *`argument`* is an object that contains iterable elements
    - e.g., *`argument`* can be a list, string, or tuple
    - If *`argument`* is a string, each character becomes a set element
      - For set of strings, pass them to the function as a list
    - If *`argument`* contains duplicates, only one of the duplicates will appear in the set

Getting the Number of and Adding Elements

- `len` function: returns the number of elements in the set
- Sets are mutable objects
- `add` method: adds an element to a set
- `update` method: adds a group of elements to a set
  - Argument must be a sequence containing iterable elements, and each of the elements is added to the set

Deleting Elements From a Set

- **`remove` and `discard` methods: remove the specified item from the set**
  - The item that should be removed is passed to both methods as an argument
  - Behave differently when the specified item is not found in the set
    - `remove` method raises a `KeyError` exception
    - `discard` method does not raise an exception
- **`clear` method: clears all the elements of the set**

Using the `for` Loop, `in`, and `not in` Operators With a Set

- **A `for` loop can be used to iterate over elements in a set**
  - General format: `for` *`item`* `in` *`set`*`:`
  - The loop iterates once for each element in the set
- **The `in` operator can be used to test whether a value exists in a set**
  - Similarly, the `not in` operator can be used to test whether a value does not exist in a set

Finding the Union of Sets

- **<u>Union of two sets</u>: a set that contains all the elements of both sets**
- **To find the union of two sets:**
  - Use the `union` method
    - Format: `*set1*.union(*set2*)`
  - Use the `|` operator
    - Format: `*set1* | *set2*`
  - Both techniques return a new set which contains the union of both sets

Finding the Intersection of Sets

- **<u>Intersection of two sets</u>: a set that contains only the elements found in both sets**
- **To find the intersection of two sets:**
  - Use the `intersection` method
    - Format: `set1.intersection(set2)`
  - Use the `&` operator
    - Format: `set1 & set2`
  - Both techniques return a new set which contains the intersection of both sets

Finding the Difference of Sets

- **Difference of two sets: a set that contains the elements that appear in the first set but do not appear in the second set**
- **To find the difference of two sets:**
  - Use the `difference` method
    - Format: `set1.difference(set2)`
  - Use the – operator
    - Format: `set1 - set2`

Finding the Symmetric Difference of Sets

- **<u>Symmetric difference of two sets</u>: a set that contains the elements that are not shared by the two sets**
- **To find the symmetric difference of two sets:**
  - Use the `symmetric_difference` method
    - Format: `set1.symmetric_difference(set2)`
  - Use the `^` operator
    - Format: `set1 ^ set2`

Finding Subsets and Supersets

- **Set A is subset of set B if all the elements in set A are included in set B**
- **To determine whether set A is subset of set B**
  - Use the `issubset` method
    - Format: `*setA*.issubset(*setB*)`
  - Use the `<=` operator
    - Format: `*setA* <= *setB*`

Finding Subsets and Supersets (cont'd.)

- **Set A is superset of set B if it contains all the elements of set B**
- **To determine whether set A is superset of set B**
  - Use the `issuperset` method
    - Format: `setA.issuperset(setB)`
  - Use the `>=` operator
    - Format: `setA >= setB`

Serializing Objects

- Serialize an object: convert the object to a stream of bytes that can easily be stored in a file
- Pickling: serializing an object

Serializing Objects (cont'd.)

- **To pickle an object:**
  - Import the `pickle` module
  - Open a file for binary writing
  - Call the `pickle.dump` function
    - Format: `pickle.dump(`*`object, file`*`)`
  - Close the file
- **You can pickle multiple objects to one file prior to closing the file**

Serializing Objects (cont'd.)

- **<u>Unpickling</u>: retrieving pickled object**
- **To unpickle an object:**
  - Import the `pickle` module
  - Open a file for binary writing
  - Call the `pickle.load` function
    - Format: `pickle.load(`*`file`*`)`
  - Close the file
- **You can unpickle multiple objects from the file**

Summary

- **This chapter covered:**
  - Dictionaries, including:
    - Creating dictionaries
    - Inserting, retrieving, adding, and deleting key-value pairs
    - `for` loops and `in` and `not in` operators
    - Dictionary methods

Summary (cont'd.)

- **This chapter covered (cont'd):**
  - Sets:
    - Creating sets
    - Adding elements to and removing elements from sets
    - Finding set union, intersection, difference and symmetric difference
    - Finding subsets and supersets
  - Serializing objects
    - Pickling and unpickling objects