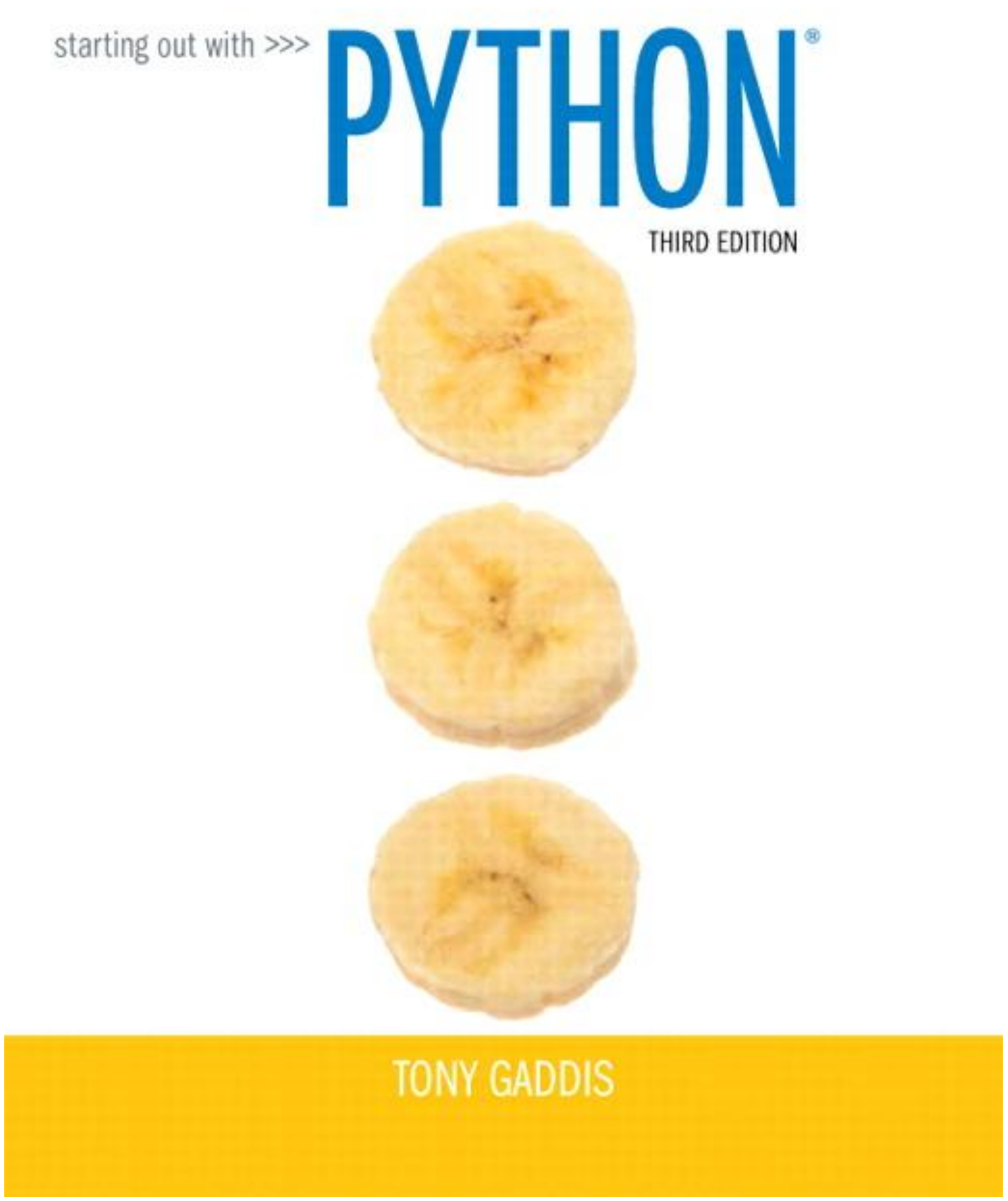
CHAPTER 9

Dictionaries and Sets



Topics

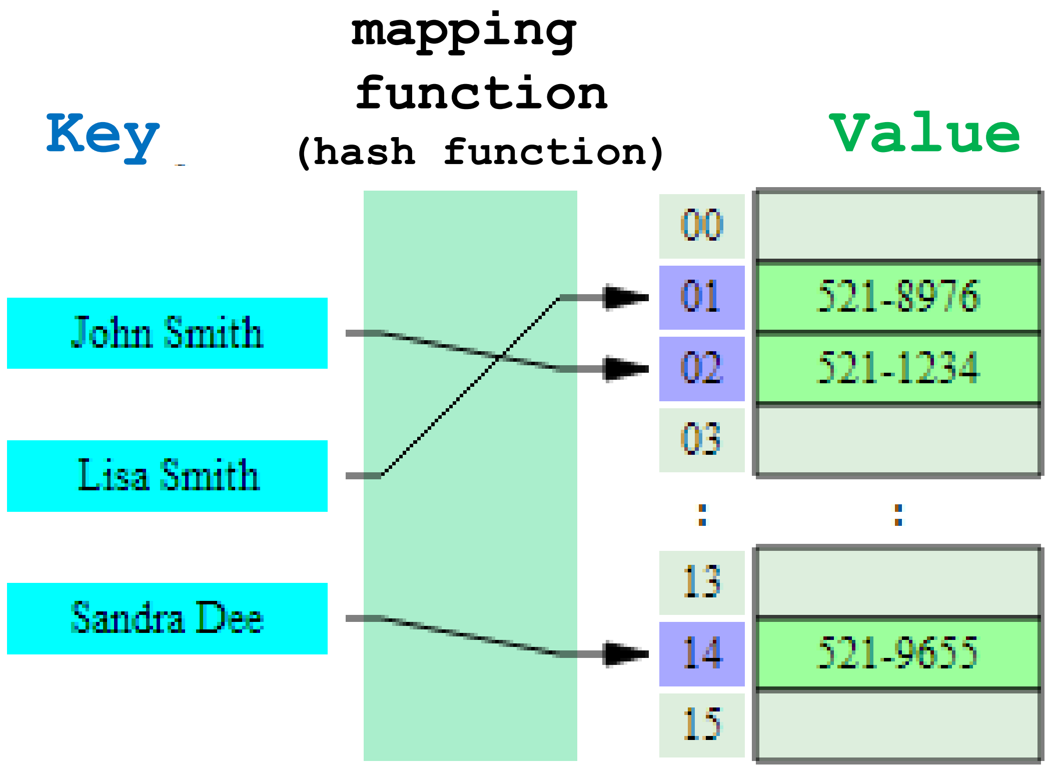
- Dictionaries
- Sets
- Serializing Objects

Dictionaries

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

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

Hash Table or Dictionary



Creating an Empty Dictionary

- **To create an empty dictionary:**
  - Use `{}`
  - Use built-in function `dict()`
  - A colon (:) separates a key and its value
  - Elements can be added to the dictionary as program executes

```
dict = {“Alice”: “2341”, “Beth”: “9102”, “Cecil”: “3258”}
```

dictionary1.py

Using `for` Loop to Iterate Over a Dictionary

- **Use a `for` loop to iterate over a dictionary keys**
  - General format: `for` *`key`* `in` *`dictionary`*`:`

```
for key in dictionary:
    print(dictionary[key])
```

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

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

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

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**

Some Dictionary Methods

- **`clear` method: deletes all the elements in a dictionary, leaving it empty**
  - Format: `dictionary.clear()`
- **`get` method: 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

- **`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

- **<u>`keys` method</u>: returns all the dictionaries keys as a sequence**
  - Format: `*dictionary*.keys()`
- **<u>`pop` method</u>: 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

- **`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

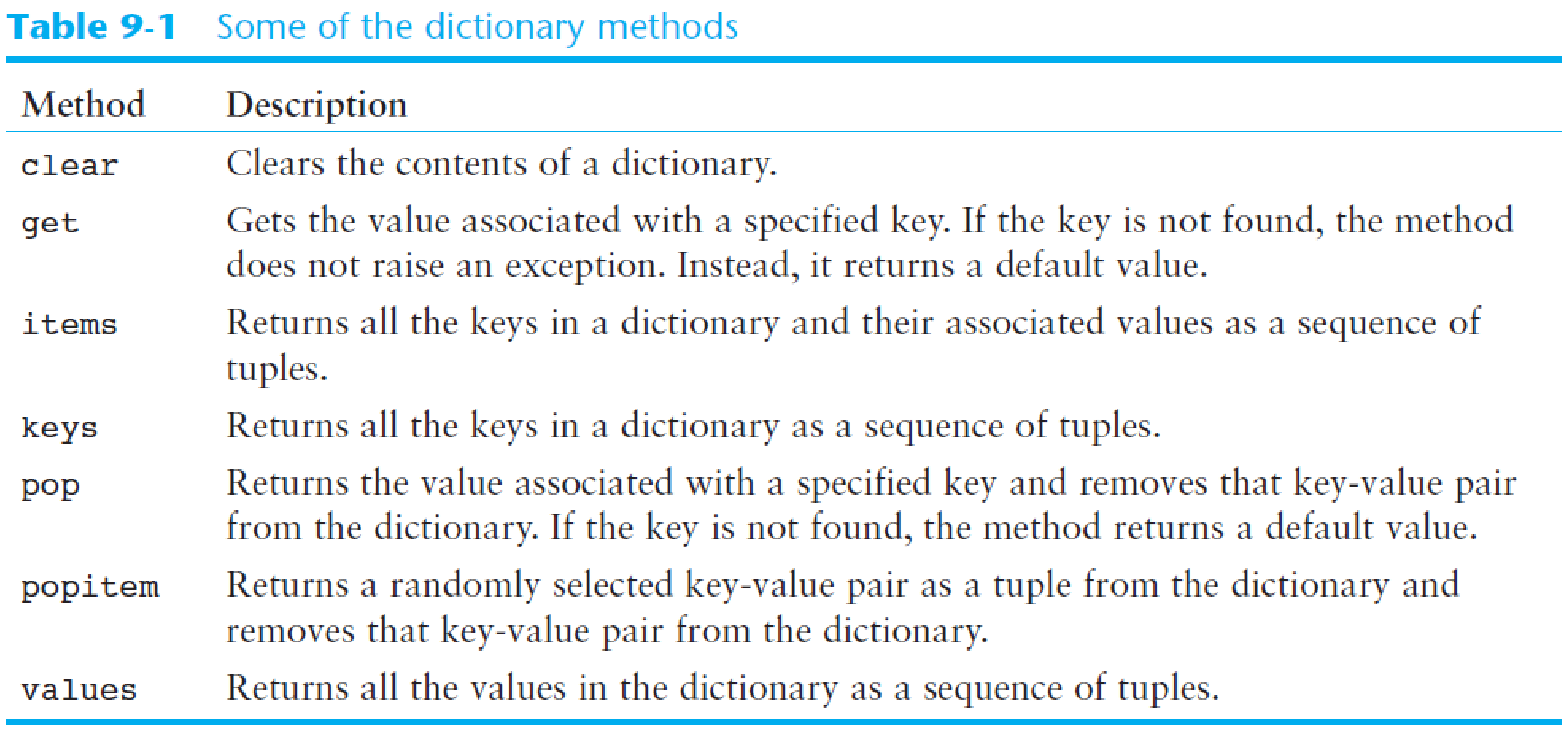


Table 9-1 Some of the dictionary methods

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.

dictionary1.py and dictionary2.py

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

- **`set` function**: **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

- **Intersection of two sets: 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

- **Symmetric difference of two sets: 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

- **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

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

Serializing Objects

- **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**

pickleDictionary.py

Serializing Objects

- **Unpickling: 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

- **This chapter covered:**
  - 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