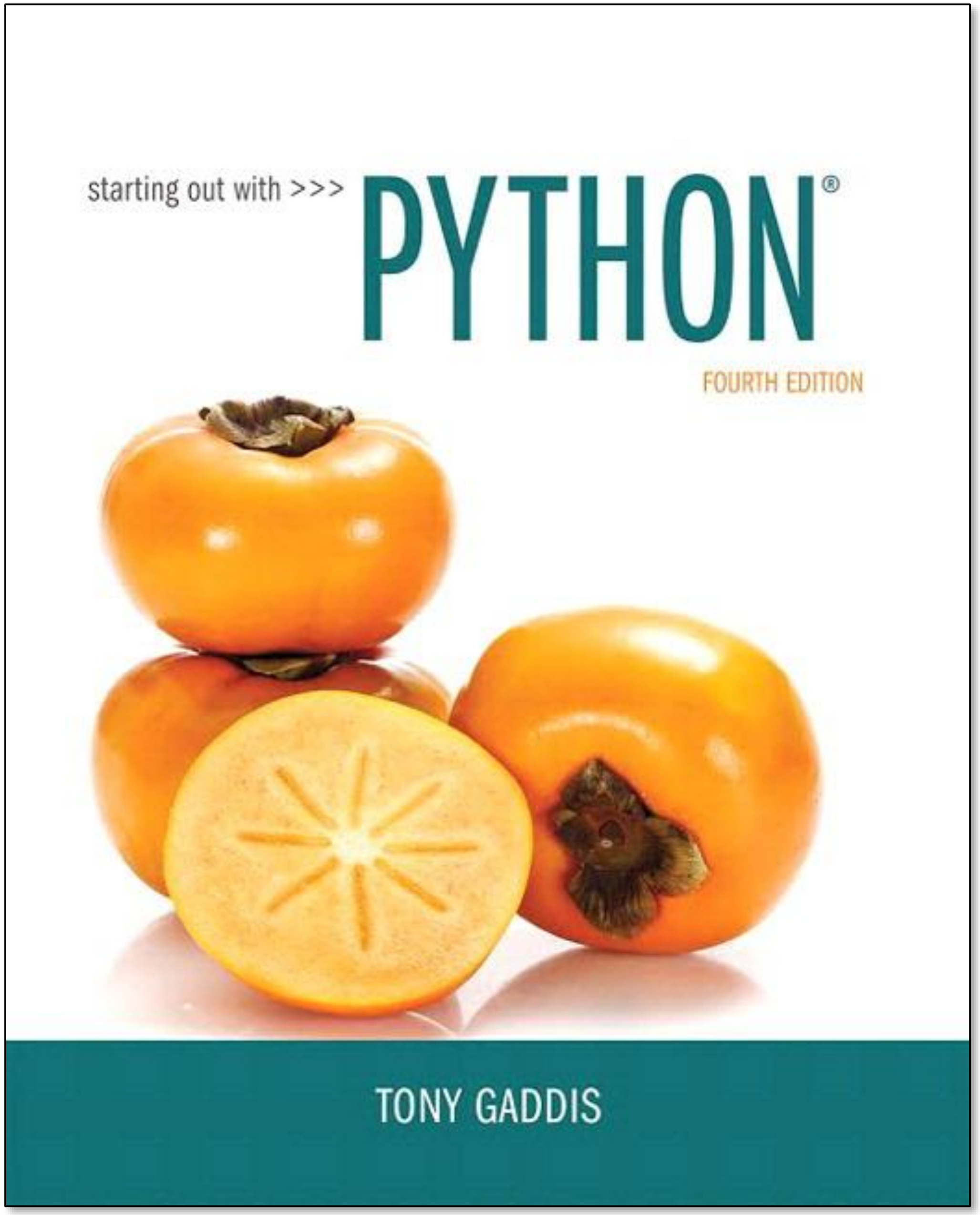
CHAPTER 9

Dictionaries and Sets



Topics

- Dictionaries
- Sets
- Serializing Objects

Dictionaries

- **<u>Dictionary</u>: object that stores a collection of data**
  - Each element consists of *<u>a key and a value</u>*
    - 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}
```

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

```
>>> phonebook = {'Cris':'555-1111','Katie':'555-2222','John':'555-3333'}
>>> phonebook
{'Cris': '555-1111', 'Katie': '555-2222', 'John': '555-3333'}
>>> phonebook['Cris']
'555-1111'
>>> phonebook['Rabieh']
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    phonebook['Rabieh']
KeyError: 'Rabieh'
>>> if 'Cris' in phonebook:
        print(phonebook['Cris'])


555-1111
```

Adding Elements to an Existing Dictionary

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

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

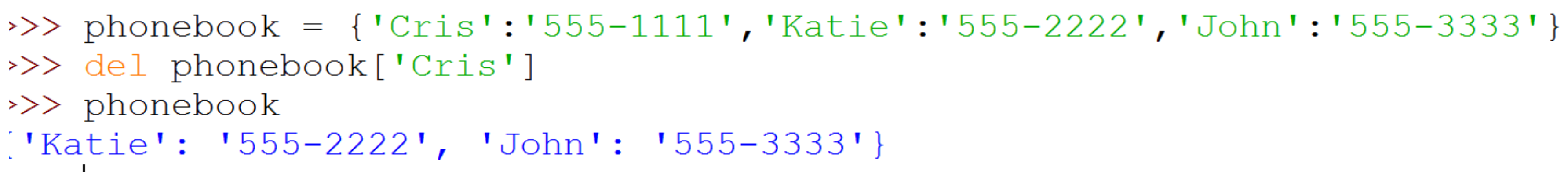
  - If key exists in the dictionary, the value associated with it will be replaced
  - You can not have duplicate keys in the dictionary

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



```
>>> phonebook = {'Cris':'555-1111','Katie':'555-2222','John':'555-3333'}
>>> del phonebook['Cris']
>>> phonebook
{'Katie': '555-2222', 'John': '555-3333'}
```

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**

```
>>> test_scores = {'Kayla':[88,90,100], 'Luis':[90,67,100], 'Ethan':[88,23,87]}
>>> kayla_score = test_scores['Kayla']
>>> kayla_score
[88, 90, 100]
>>> diff dictionary = {1:'John', '1':'Mryam'}
```

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

- **To create an empty dictionary:**
  - Use `{}`
  - Use built-in function `dict()`
    - `Phonebook = 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`*`:`

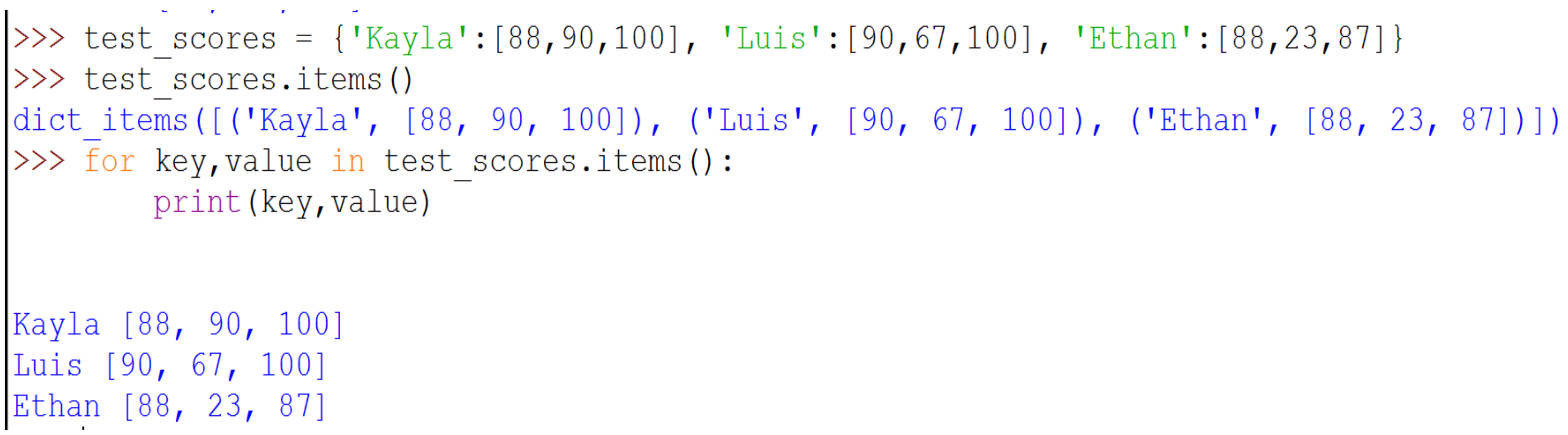
```
>>> test_scores = {'Kayla':[88,90,100], 'Luis':[90,67,100], 'Ethan':[88,23,87]}
>>> for name in test_scores:
        print (name,':',test_scores[name])


Kayla : [88, 90, 100]
Luis : [90, 67, 100]
Ethan : [88, 23, 87]
```

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
    - Does not raise `KeyError` exception

Some Dictionary Methods (cont'd.)

- **`items` method: returns all the dictionaries keys and associated values**
  - Format: *`dictionary`*`.items()`
  - Returns 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



```
>>> test_scores = {'Kayla':[88,90,100], 'Luis':[90,67,100], 'Ethan':[88,23,87]}
>>> test_scores.items()
dict_items([('Kayla', [88, 90, 100]), ('Luis', [90, 67, 100]), ('Ethan', [88, 23, 87])])
>>> for key,value in test_scores.items():
        print(key,value)


Kayla [88, 90, 100]
Luis [90, 67, 100]
Ethan [88, 23, 87]
```

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
  - Similar to get method, however get does not remove the item

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 (cont'd.)

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.

```
>>> test_scores = {'Kayla':[88,90,100], 'Luis':[90,67,100], 'Ethan':[88,23,87]}
>>> test_scores.keys()
dict_keys(['Kayla', 'Luis', 'Ethan'])
>>> test_scores.values()
dict_values([[88, 90, 100], [90, 67, 100], [88, 23, 87]])
>>> test_scores.get('Cris','Entry Not found')
'Entry Not found'
>>> test_scores.get('Kayla','Entry Not found')
[88, 90, 100]
>>> test_scores.items()
dict_items([('Kayla', [88, 90, 100]), ('Luis', [90, 67, 100]), ('Ethan', [88, 23, 87])])
>>> test_scores.pop('Cris', 'Not found')
'Not found'
>>> test_scores.pop('Kayla', 'Not found')
[88, 90, 100]
>>> test_scores
{'Luis': [90, 67, 100], 'Ethan': [88, 23, 87]}
>>> test_scores.popitem()
('Ethan', [88, 23, 87])
>>> test_scores
{'Luis': [90, 67, 100]}
```

Case Study

card_dealer.py

- Simulate deck of cards game
- Each card a numeric value
- Jacks, queen and kings are valued 10
- The use should pick number of cards (n)
- The program picks (n) random cards and displays their total sum

Sets

- **Set: 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

Set takes only one Argument

```
>>> set1 = set('John Sina')
>>> set1
{'S', 'J', 'h', 'a', ' ', 'i', 'n', 'o'}
>>> set2 = set(['1','2','3'])
>>> set2
{'1', '3', '2'}
>>> set3 = set(('1','2','3'))
>>> set3
{'1', '3', '2'}
```

Getting the Number of and Adding Elements

- `len` function: returns the number of elements in the set
- Sets are mutable objects
  - Can ad or remove items
- `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

```
>>> set3
{'3', 4, 5, 6, '2', '1'}
>>> set3.add(8)
>>> set3.add('a')
>>> set3.add('aaa')
>>> set3.add('abc')
>>> set3
{'abc', '3', 4, 5, 6, 8, 'a', '2', 'aaa', '1'}
>>> set3.update('abc')
>>> set3
{'abc', '3', 4, 5, 6, 8, 'a', '2', 'aaa', '1', 'b', 'c'}
```

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

```
>>> set1 = set([1,2,3,4,5,6,7])
>>> set2 = set([5,6,7,8,9,10,11])
>>> set3 = set1.union(set2)
>>> set3
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
>>> set4 = set1.intersection(set2)
>>> set4
{5, 6, 7}
>>> set3  = set1 | set2
>>> set3
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
>>> set4 = set1 & set2
>>> set4
{5, 6, 7}
```

Finding the Difference of Sets

- **<u>Difference of two sets</u>: 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 (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.)

simple_pickle.py

pickle_objects.py

- **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 reading
  - Call the `pickle.load` function
    - Format: `pickle.load(`*`file`*`)`
  - Close the file
- **You can unpickle multiple objects from the file**

simple_unpickle.py

unpickle_objects.py

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