

Dictionaries and Sets

Topics

- Dictionaries
- Sets
- Serializing Objects

2

Dictionaries

- Dictionary: 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}
```

3

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

4

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

5

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

6

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

7

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:`

8

Some Dictionary Methods (1 of 5)

- 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

9

Some Dictionary Methods (2 of 5)

- 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

10

Some Dictionary Methods (3 of 5)

- 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

11

Some Dictionary Methods (4 of 5)

- popitem method: Returns, as a tuple, the key-value pair that was last added to the dictionary. The method also removes the 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

12

Some Dictionary Methods (5 of 5)

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, as a tuple, the key-value pair that was last added to the dictionary. The method also removes the key-value pair from the dictionary.
values	Returns all the values in the dictionary as a sequence of tuples.

13

Dictionary Comprehensions (1 of 6)

- Dictionary comprehension: an expression that reads a sequence of input elements and uses those input elements to produce a dictionary

14

Dictionary Comprehensions (2 of 6)

- Example: create a dictionary in which the keys are the integers 1 through 4 and the values are the squares of the keys

Using a for loop

```
>>> numbers = [1, 2, 3, 4]
>>> squares = {}
>>> for item in numbers:
...     squares[item] = item**2
...
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16}
>>>
```

Using a dictionary comprehension

```
>>> squares = {item:item**2 for item in numbers}
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16}
>>>
```

15

Dictionary Comprehensions (3 of 6)

```
squares = {item:item**2 for item in numbers}
           └───┬───┘ └───┬───┘
           Result Expression Iteration Expression
```

- The iteration expression iterates over the elements of numbers
- Each time it iterates, the target variable item is assigned the value of an element
- At the end of each iteration, an element containing item as the key and item**2 as the value is added to the new dictionary

16

Dictionary Comprehensions (4 of 6)

- Example: You have an existing list of strings. Create a dictionary in which the keys are the strings in the list, and the values are the lengths of the strings

```
>>> names = ['Jeremy', 'Kate', 'Peg']
>>> str_lengths = {item:len(item) for item in names}
>>> str_lengths
{'Jeremy': 6, 'Kate': 4, 'Peg': 3}
>>>
```

17

Dictionary Comprehensions (5 of 6)

- Example: making a copy of a dictionary

```
>>> dict1 = {'A':1, 'B':2, 'C':3}
>>> dict2 = {k:v for k,v in dict1.items()}
>>> dict2
{'A': 1, 'B': 2, 'C': 3}
>>>
```

18

Dictionary Comprehensions (6 of 6)

- You can use an `if` clause in a dictionary comprehension to select only certain elements of the input sequence
 - Example: A dictionary contains cities and their populations as key-value pairs. Select only the cities with a population greater than 2 million

```
>>> populations = {'New York': 8398748, 'Los Angeles': 3990456,
...               'Chicago': 2705994, 'Houston': 2325502,
...               'Phoenix': 1660272, 'Philadelphia': 1584138}
>>> largest = {k:v for k,v in populations.items() if v > 2000000}
>>> largest
{'New York': 8398748, 'Los Angeles': 3990456, 'Chicago': 2705994,
 'Houston': 2325502}
>>>
```

19

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

20

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

21

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

22

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

23

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

24

Finding the Union of Sets

- Union of two sets: 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

25

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

26

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`

27

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`

28

Finding Subsets and Supersets (1 of 2)

- 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`

29

Finding Subsets and Supersets (2 of 2)

- 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`

30

Set Comprehensions (1 of 4)

- Set comprehension: a concise expression that creates a new set by iterating over the elements of a sequence
- Set comprehensions are written just like list comprehensions, except that a set comprehension is enclosed in curly braces (`{}`) instead of brackets (`[]`)

31

Set Comprehensions (2 of 4)

- Example: making a copy of a set

```
>>> set1 = set([1, 2, 3, 4, 5])
>>> set2 = {item for item in set1}
>>> set2
{1, 2, 3, 4, 5}
>>>
```

32

Set Comprehensions (3 of 4)

- Example: creating a set that contains the squares of the numbers stored in another set

```
>>> set1 = set([1, 2, 3, 4, 5])
>>> set2 = {item**2 for item in set1}
>>> set2
{1, 4, 9, 16, 25}
>>>
```

33

Set Comprehensions (4 of 4)

- Example: copying the numbers in a set that are less than 10

```
>>> set1 = set([1, 20, 2, 40, 3, 50])
>>> set2 = {item for item in set1 if item < 10}
>>> set2
{1, 2, 3}
>>>
```

34

Serializing Objects (1 of 3)

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

35

Serializing Objects (2 of 3)

- 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

36

Serializing Objects (3 of 3)

- 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

37

Summary (1 of 2)

- 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

38

Summary (2 of 2)

- 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

39