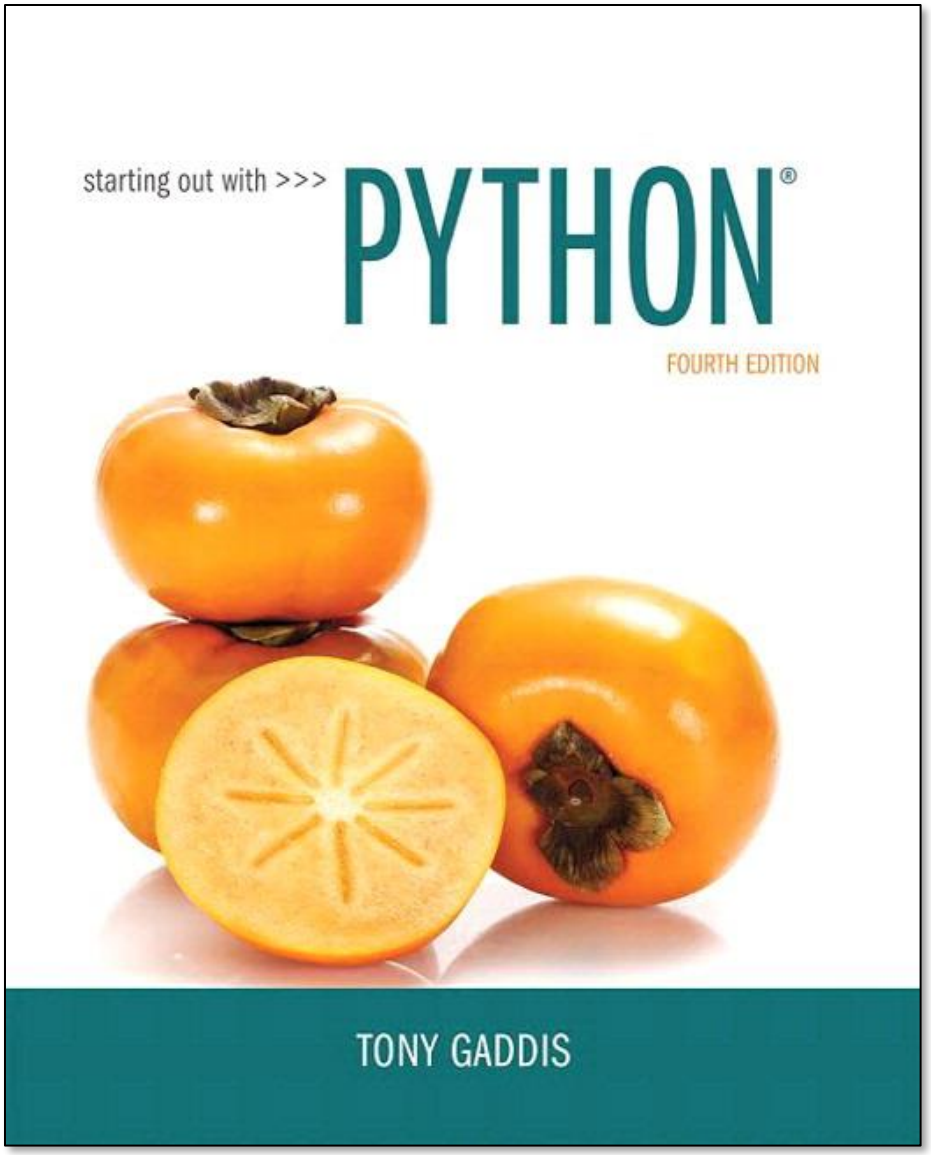
CHAPTER 9

Dictionaries and Sets



Dr. Bita Shadgar

Topics

- Sets
- Dictionaries

Learning Outcomes

- At the end of this week the students must be able to:
  - Create sets and add elements to and remove elements from sets
  - Apply union, intersection, difference and symmetric difference on sets
  - Find subsets and supersets
  - Create dictionaries and insert, retrieve, add, and delete key-value pairs in a dictionary
  - Iterate on a dictionaries using a `for` loops
  - Apply dictionary methods

Sets

- **Set:** like a list, it is an object that stores a collection of data in same way as mathematical set, but
  - All items must be unique
  - Set is unordered
  - Elements can be of different data types

Creating a Set

- `set`: 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
    - If *`argument`* contains duplicates, only one of the duplicates will appear in the set

```
>>> x = set()
>>> x = set("hello")
>>> x
{'e', 'l', 'o', 'h'}
>>> y = set(["hello", "world", 3])
>>> y
{3, 'hello', 'world'}
>>> len(y)
3
>>>
```

Getting the Number of 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

Example – Set's Method

```
>>> y
{3, 'hello', 'world'}
>>> len(y)
3
>>> y.add('new item')
>>> y
{3, 'new item', 'hello', 'world'}
>>> y.update([3,4,5])
>>> y
{3, 4, 5, 'hello', 'world', 'new item'}
```

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

`in`, and `not in` Operators

- 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

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

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

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`*

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`*

Example – Set Methods

```
>>> x = set((1,2,3,4,5))
>>> y = set([8,7,6,5,4])
>>> y.union(x)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> y.intersection(x)
{4, 5}
>>> y.difference(x)
{8, 6, 7}
>>> y.symmetric_difference(x)
{1, 2, 3, 6, 7, 8}
```

Subsets

- 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`*

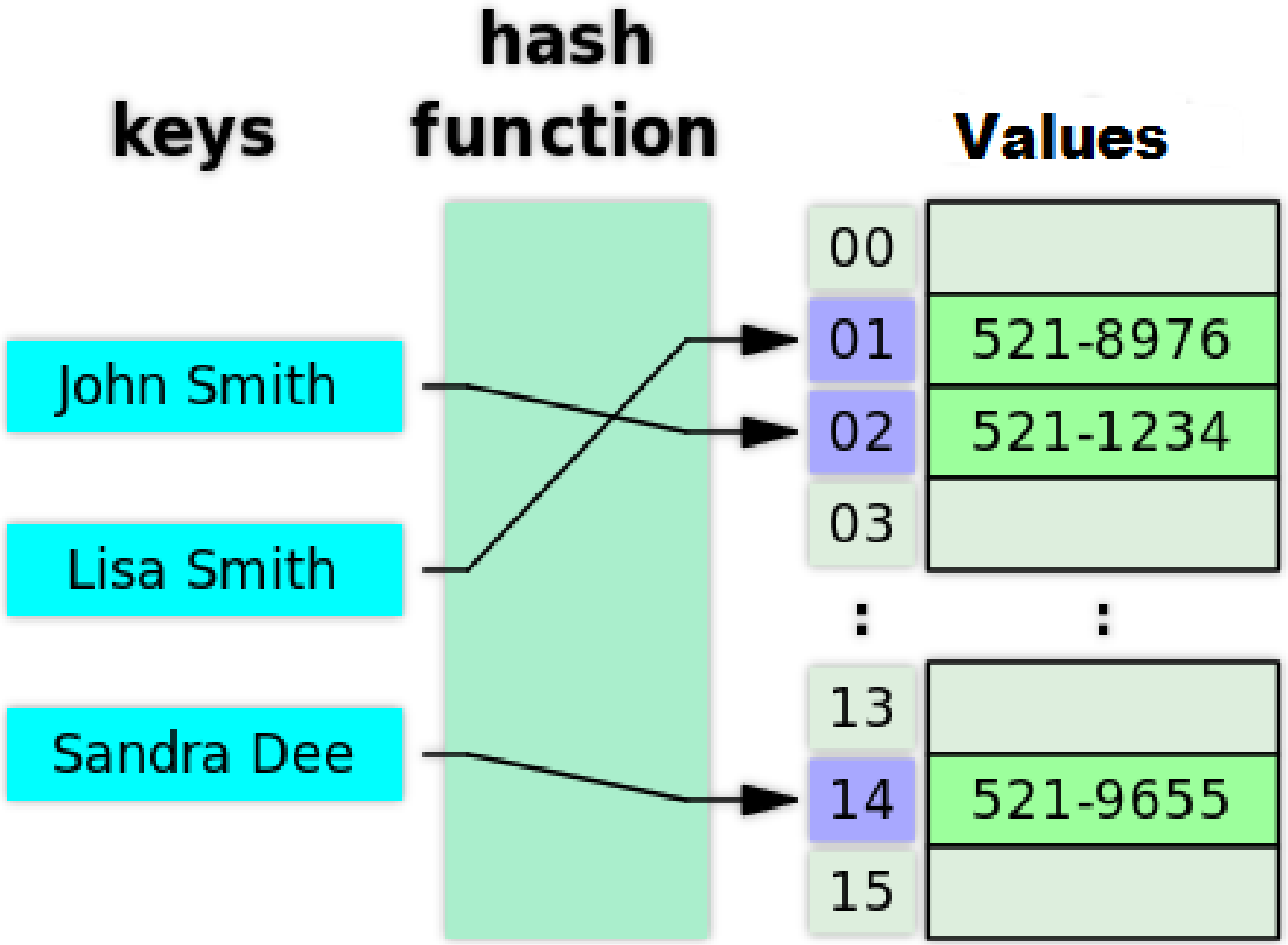
Supersets

- 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`*

Dictionary

- Similar to lists but instead of accessing stored values by the index, we access them by the key
  - Calling a friend using phone book
  - Name of your friend is the key and number is the value (“Lisa Smith”, “521-8976”)
  - If we had a list of phone number, we had to know what is the index of Lisa’s phone number (eg. index 1)
- Dictionary is a mapping between a set of (key, value) and indices of a table

A Small Phone Book



Dictionaries

- **Dictionary:** object that stores a collection of data
  - To retrieve a specific value, use the key associated with it
  - Format for creating a dictionary

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

Creating Dictionary

- Each element consists of a *key* and a *value*
  - Often referred to as *mapping* of key to value
  - Key must be an immutable object
- Python format:

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

```
>>> marbles = {'red': 34, 'green': 30}  #create a dictionary with initial items
>>> marbles
{'red': 34, 'green': 30}
```

- To create an empty dictionary:

```
>>> marbles = {}   # create an empty dictionary using {}
>>> marbles = dict() # create an empty dictionary using a built-in fun.
```

Retrieving a Value from a Dictionary

- General format : *dictionary*[*key*]
  - If `key` is in the dictionary, associated value is returned, otherwise, `KeyError` exception is raised

```
>>> marbles
{'red': 34, 'yellow': 29, 'green': 30}
>>> marbles['green']  #get a value by its key
30
>>> marbles['blue']    #get a value of a key that doesn't exist
KeyErr: 'blue'
```

How to prevent `KeyError` exceptions

- `in` (`not in`) operator is used to check whether or not a key is available

```
>>> mydict
{'age': 23, 'course': 183}
>>> x = 'age'
>>> if x in mydict:
. . .       print (mydict[x])
23
>>> y = 'name'
>>> if y in mydict:  # key does not exist, none is returned
. . .       print (mydict[y])
>>>
```

```
>>> mydict
{'age': 23, 'course': 183}
>>> 'age' in mydict
True
>>> 'name' in mydict
False
```

Adding Elements to a Dictionary

- Dictionaries are mutable objects
- In general, the order of items in a dictionary is not predictable
- To add a new key-value pair:

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

```
>>> marbles
{'red': 34, 'green': 30}
>>> marbles['yellow'] = 29    #add an item
>>> marbles
{'red':34, 'yellow': 29, 'green': 30}
```

Keys of a Dictionary

- Keys are unique
- If key exists in the dictionary, the value associated with it will be updated

```
>>> marbles
{'red': 34, 'yellow': 29, 'green': 30}
>>> marbles['red'] = 10      #update an item
>>> marbles
{'red': 10, 'green': 30, 'yellow': 29}
>>> marbles['green'] = marbles['yellow'] + 5  #overwrite an item
{'red': 10, 'green': 34, 'yellow': 29}
```

Deleting Elements From a Dictionary

- To delete a key-value pair:

```
del dictionary[key]
```

  - If key is not in the dictionary, `KeyError` exception is raised
  - Again you can use in operator to prevent the exception

```
>>> if x in mydict:
...     del mydict[x]
```

Getting the Number of Elements

- **`len` function:** used to obtain number of elements in a dictionary

```
>>> marbles
{‘red':34, ‘yellow’: 29, ‘green': 30}
>>> size = len(marbles)
>>> print(size)
3
```

Using `for` Loop to Iterate Over a Dictionary

- Use a `for` loop to iterate over a dictionary
  - General format:

    `for` *`key`* `in` *`dictionary`*`:`

```
>>> marbles
{'red': 10, 'green': 30, 'yellow': 29}
>>> for y in marbles:  # iterate over keys
. . .       print (y, marbles[y])
red  10
green  30
yellow  29
>>>
```

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

```
>>> for k, v in marbles.items():
...     print (k, v)
```

Some Dictionary Methods (cont'd.)

- **`keys`**: returns all the dictionaries keys as a sequence
  - Format: *`dictionary`*`.keys()`
- **`values`**: 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.)

- **`pop`**: 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
- **`popitem`**: 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

Example – Dictionary's Method

```
>>> marbles = {"red": 34, "green": 30, "brown": 31 }
>>> marbles.get("red")        #get a value by its key, or None if it doesn't exist
34
>>> marbles.update({"orange": 24, "blue": 23}) #add several items
>>> marbles
{"red": 34, "orange": 24, "green": 30, "brown": 31, "blue": 23}
>>> marbles.keys()                          #list all the keys in the dictionary
["red", "orange", "green", "brown", "blue"]
>>> marbels.values()                        #list all the values in the dictionary
[34, 24, 30, 31, 23]
>>> marbles.items()                         #return a list of the items
[("red", 34) , ("orange", 24), ("green", 30), ("brown", 31), ("blue", 23)]
>>> marbels.pop("red")                              #remove an item
>>> marbles
{"orange": 24, "green": 30, "brown": 31, "blue": 23}
```

Example – Counting Names

```
names = ['Tom', 'John', 'Tom', 'Adam', 'John']
counts = dict()
for name in names:
    if name not in counts:
        counts[name] = 1   #add new item
    else:
        counts[name] = counts[name] + 1    #update an item
print counts
```

{'Tom': 2, 'Adam': 1, 'John': 2}

- Improved version of for loop

```
for name in names:
    counts[name] = counts.get(name, 0) + 1
```

Summary

- Dictionaries, including:
  - Creating dictionaries
  - Inserting, retrieving, adding, and deleting key-value pairs
  - `for` loops and `in` and `not in` operators
  - Dictionary methods
- Sets, including :
  - Creating sets
  - Adding elements to and removing elements from sets
  - Finding set union, intersection, difference and symmetric difference
  - Finding subsets and supersets

More Practice

- Check out review questions in chapter 9 of the textbook including :
  - Multiple Choices
  - True or False
  - Short Answer
  - Algorithm WorkBench
  - Programming Exercises (1, 3, 4, 5, 8)