CITY UNIVERSITY
LONDON

Introduction to programming using Python

Session 6-1

Matthieu Choplin

matthieu.choplin1@city.ac.uk

http://moodle.city.ac.uk/

Objectives

- To use tuples as immutable lists
- To use sets for storing and fast accessing non-duplicated elements
- To understand the performance differences between sets and lists
- To store key/value pairs in a dictionary and access value using the keys

CITY UNIVERSITY
LONDON

Tuples

- Tuples are like lists except they are **immutable**. Once they are created, their contents cannot be changed.
- If the contents of a list in your application do not change, you should use a tuple to prevent data from being modified accidentally. Furthermore, tuples are more efficient than lists.

CITY UNIVERSITY LONDON

Creating Tuples

- With brackets `(` and `)`

```
t1 = () # Create an empty tuple
t2 = (1, 3, 5)
```

- By converting a list (comprehension here) into a tuple

```
t3 = tuple([2 * x for x in range(1, 5)])
```

- By converting a string into a tuple

```
t4 = tuple("abac")
```

Tuples -- len(), max(), min(), [] index

- Tuples can be used like lists except they are immutable

```
tuple2 = tuple([7, 1, 2, 23, 4, 5]) # Create a tuple from a list
print(tuple2)

print("length is", len(tuple2)) # Use function len
print("max is", max(tuple2)) # Use max
print("min is", min(tuple2)) # Use min
print("sum is", sum(tuple2)) # Use sum

print("The first element is", tuple2[0]) # Use indexer
```

CITY UNIVERSITY
LONDON

Tuples -- +, *, [:] slice, in

```
tuple1 = ("green", "red", "blue") # Create a tuple
tuple2 = tuple([7, 1, 2, 23, 4, 5]) # Create a tuple from a list
tuple3 = tuple1 + tuple2 # Combine 2 tuples
print(tuple3)
tuple3 = 2 * tuple1 # Multiply a tuple
print(tuple3)
print(tuple2[2 : 4]) # Slicing operator
print(tuple1[-1])
print(2 in tuple2) # in operator
for v in tuple1:
    print(v, end = " ")
print()
```

CITY UNIVERSITY LONDON

Tuples -- +, *, [:] slice, in

```
tuple1 = ("green", "red", "blue")
tuple2 = tuple([7, 1, 2, 23, 4, 5])
list1 = list(tuple2) # Obtain a list from a tuple
list1.sort()
tuple4 = tuple(list1)
tuple5 = tuple(list1)
print(tuple4)
print(tuple4 == tuple5) # Compare two tuples
```

CITY UNIVERSITY
LONDON

Sets

- Sets are like lists to store a collection of items. Unlike lists, the elements in a set are:
  - **unique**
  - **not placed in any particular order**
- If your application does not care about the order of the elements, using a set to store elements is more efficient than using lists.
- The syntax for sets is braces {}.

Creating Sets

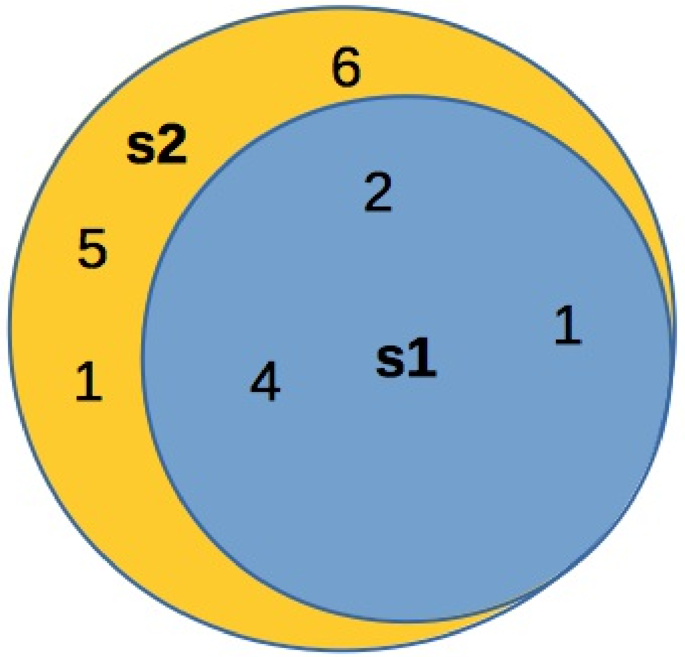
```
s1 = set() # Create an empty set
s2 = {1, 3, 5} # Create a set with three elements
s3 = set((1, 3, 5)) # Create a set from a tuple
# Create a set from a list (comprehension here)
s4 = set([x * 2 for x in range(1, 10)])
# Create a set from a string
s5 = set("abac") # s5 is {'a', 'b', 'c'}
```

Manipulating and Accessing Sets

```
s1 = {1, 2, 4}
s1.add(6)
print(s1) #  {1, 2, 4, 6}
s1.remove(4)
print(s1) #  {1, 2, 6}
```

Subset and Superset

```
s1 = {1, 2, 4}
s2 = {1, 4, 5, 2, 6}
s1.issubset(s2) # s1 is a subset of s2, print True
s2.issuperset(s1) # s2 is a superset of s1, print False
```



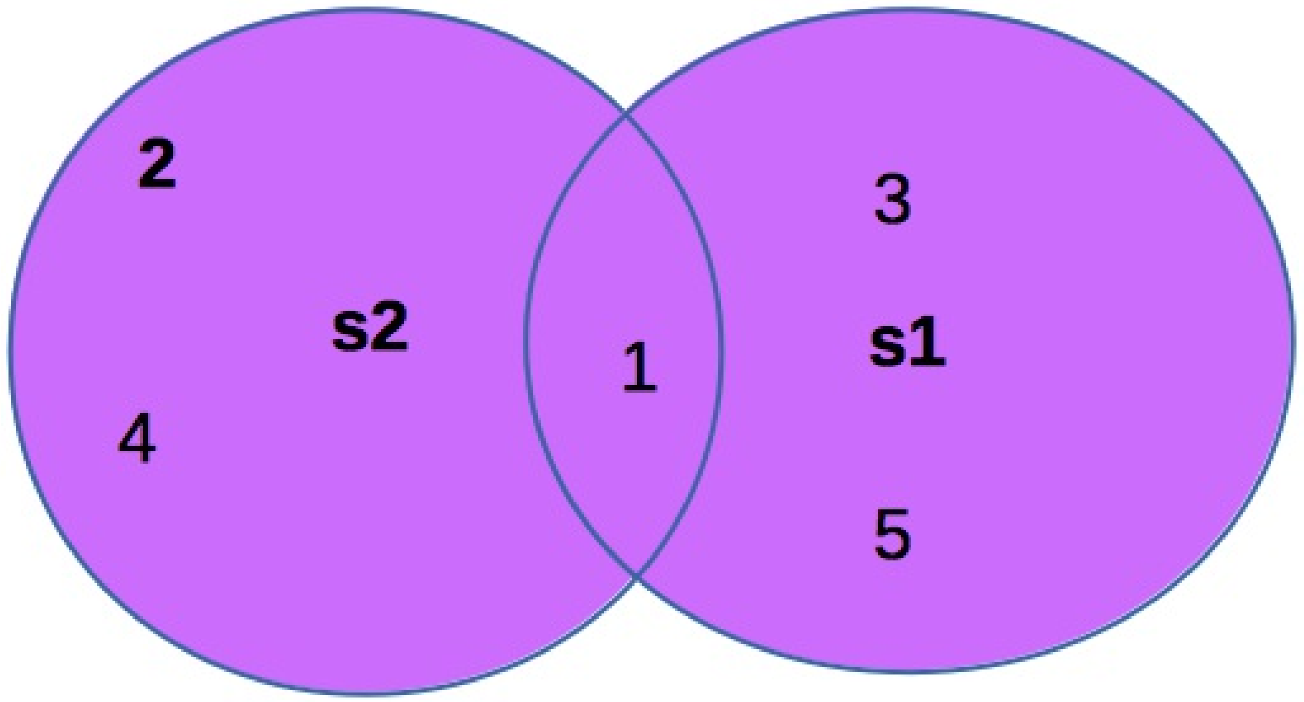
CITY UNIVERSITY LONDON

Equality Test

```
s1 = {1, 2, 4}
s2 = {1, 4, 2}
s1 == s1 #  True
s2 != s1 #  False
```

Set Operations (union, |)

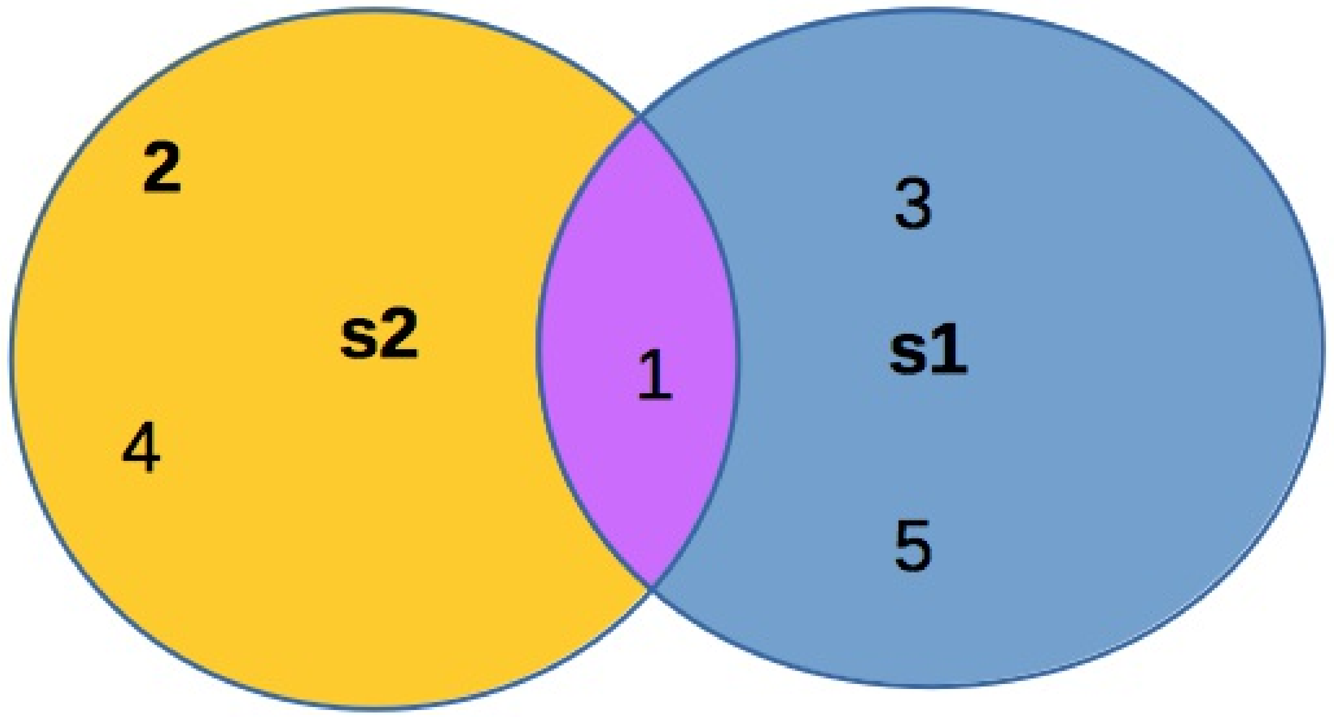
```
s1 = {1, 2, 4}
s2 = {1, 3, 5}
s1.union(s2) #  {1, 2, 3, 4, 5}
s1 | s2 # equivalent of s1.union(s2)
```



CITY UNIVERSITY
LONDON

Set Operations (intersection, &)

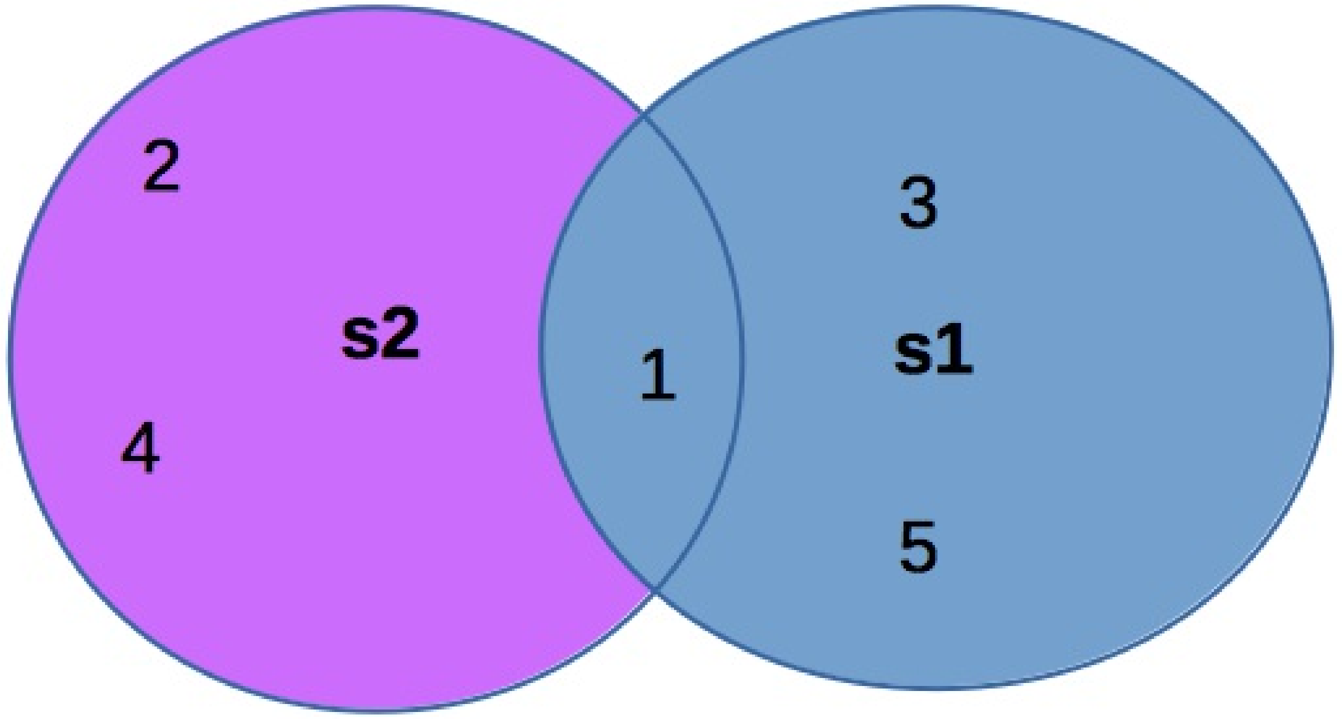
```
s1 = {1, 2, 4}
s2 = {1, 3, 5}
s1.intersection(s2) # {1}
s1 & s2 #  equivalent of s1.intersection(s2)
```



CITY UNIVERSITY LONDON

Set Operations (difference, -)

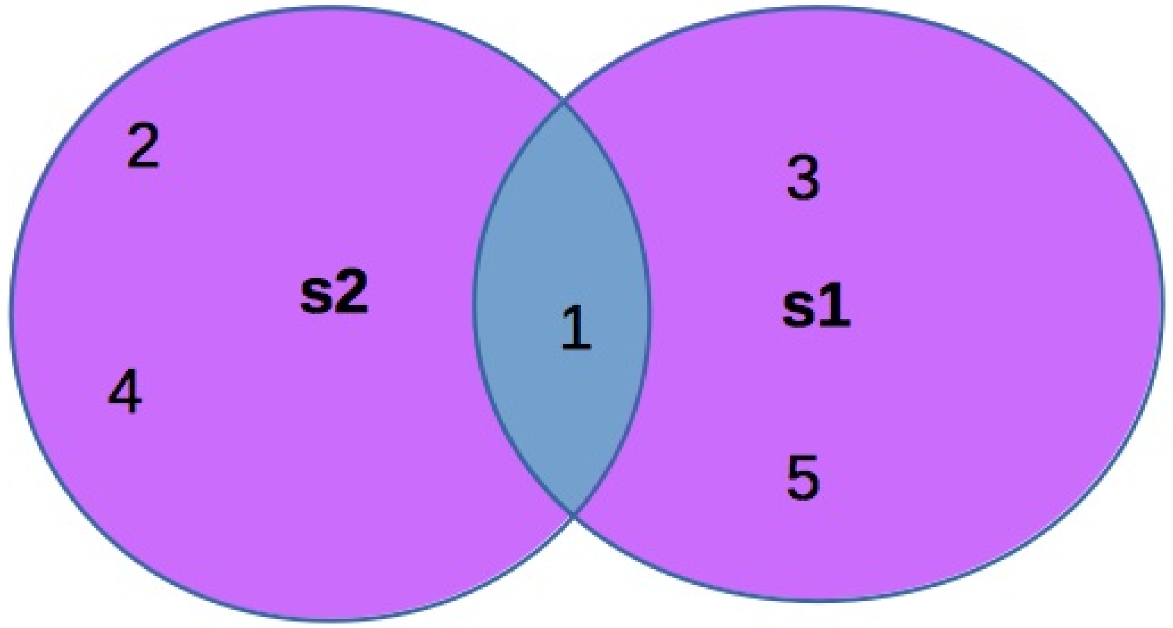
```
s1 = {1, 2, 4}
s2 = {1, 3, 5}
s1.difference(s2) # {2, 4}
s1 - s2 #  equivalent of s1.difference(s2)
```



CITY UNIVERSITY LONDON

Set Operations (symetric_difference, ^)

```
s1 = {1, 2, 4}
s2 = {1, 3, 5}
s1.symmetric_difference(s2) # {2, 3, 4, 5}
s1 ^ s2 #  equivalent of s1.symmetric_difference(s2)
```



Examples

Usage of a set SetDemo.py

Set and List performance compared:

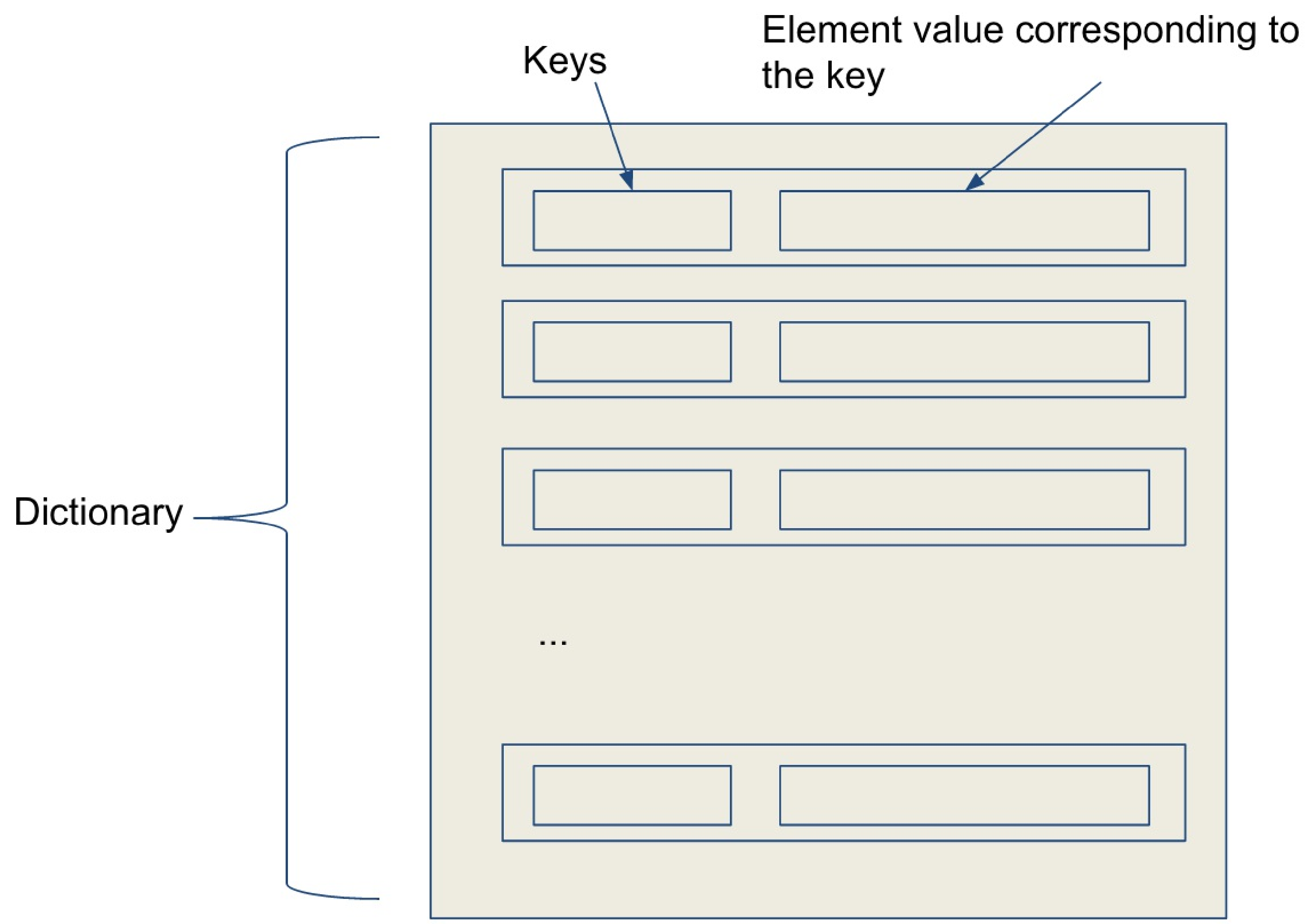
- using the time library: SetListPerformanceTest.py

CITY UNIVERSITY LONDON

Dictionary

- Why dictionary?
- Suppose your program stores a million students and frequently searches for a student using the social security number. An efficient data structure for this task is the dictionary. A dictionary is a collection that stores the elements along with the keys. The keys are like an indexer.

Key/value pairs



CITY UNIVERSITY LONDON

Creating a dictionary

```
dictionary = {} # Create an empty dictionary
dictionary = {"john":40, "peter":45}}
```

Equivalent to:

```
dictionary = dict()
dictionary =dict(john=40, peter=45)
```

Adding/Modifying Entries

To add an entry to a dictionary, use **dictionary[key] = value**

```
>>> dictionary["susan"] = 50
>>> print(dictionary)
{'john': 40, 'susan': 50, 'peter': 45}
```

CITY UNIVERSITY LONDON

Deleting Entries

To delete an entry from a dictionary, use **del dictionary[key]**

```
>>> del dictionary[“susan”]
>>> print(dictionary)
{'john': 40, 'peter': 45}
```

Looping Entries

```
for key in dictionary:
 print(key + ":" + str(dictionary[key]))
```

CITY UNIVERSITY LONDON

The len and in operators

len(dictionary) returns the number of the elements in the dictionary

```
>>> dictionary = {"john":40, "peter":45}
>>> "john" in dictionary
True
>>> "johnson" in dictionary
False
>>> len(dictionary)
2
```

The dictionary methods

Methods	Meaning
list(dictionary.keys()): list	Returns a dict_keys type of object, that you can convert in a sequence of values with list(dictionary.keys())
list(dictionary.values()): list	Returns a dict_values type of object, that you can convert with list(dictionary.values())
list(dictionary.items()): tuple	Returns a dict_items type of object, that you can convert in a sequence of tuples (key, value) with list(dictionary.items()).
clear(): None	Deletes all entries.
get(key): value	Returns the value for the key.
pop(key): value	Removes the entry for the key and returns its value

Exercise: Guess the capital

- Write a program that prompts the user to enter a capital for a random country.
- Upon receiving the user input, the program reports whether the answer is correct.
- The countries and their capitals are stored in a dictionary in this file (import it to use).
- The user’s answer is not case sensitive.

- Solution

Case Studies: Occurrences of Words

- This case study writes a program that counts the occurrences of words in a text file and displays the words and their occurrences in alphabetical order of words. The program uses a dictionary to store an entry consisting of a word and its count. For each word, check whether it is already a key in the dictionary. If not, add to the dictionary an entry with the word as the key and value 1. Otherwise, increase the value for the word (key) by 1 in the dictionary
- See the program CountOccurenceOfWords.py