CITY
UNIVERSITY OF LONDON
EST 1894

Introduction to programming with Python

Session 6-1

Objectives

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

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.

Creating Tuples

- With brackets (and)

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

- By converting a list (in this case comprehension) 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

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

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

Tuples – +, *, [:] slice, in

```
tuple1 = ("green", "red", "blue") # Create a tuple
tuple2 = tuple([7, 1, 2, 23, 4, 5]) # Create a tuple f
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()
```

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
```

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

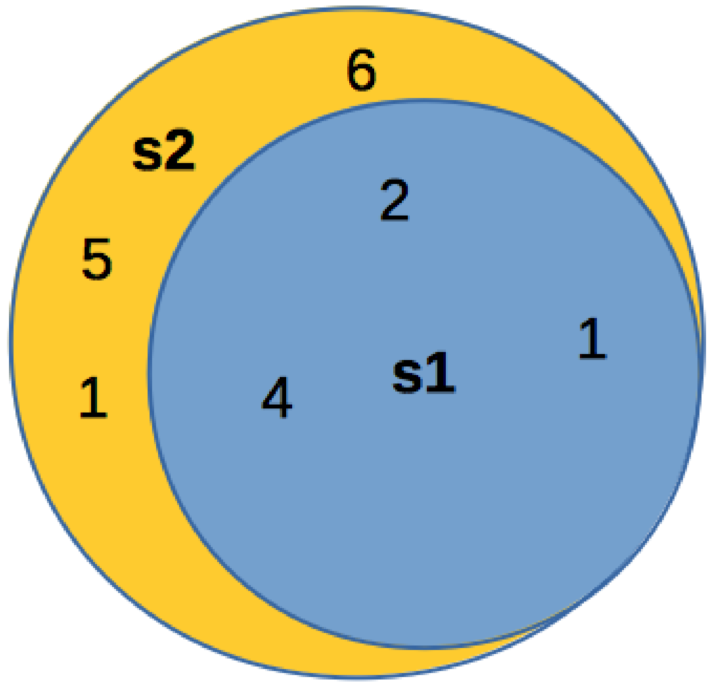
```
s1 = set() # Create an empty set
s2 = {1, 3, 5} # Create a set with three elements
s3 = set((1, 3, 5)) # Create a set from a tuple
# Create a set from a list (comprehension here)
s4 = set([x * 2 for x in range(1, 10)])
# Create a set from a string
s5 = set("abac") # s5 is {'a', 'b', 'c'}
```

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, prints True
s2.issuperset(s1) # s2 is a superset of s1, prints True
```

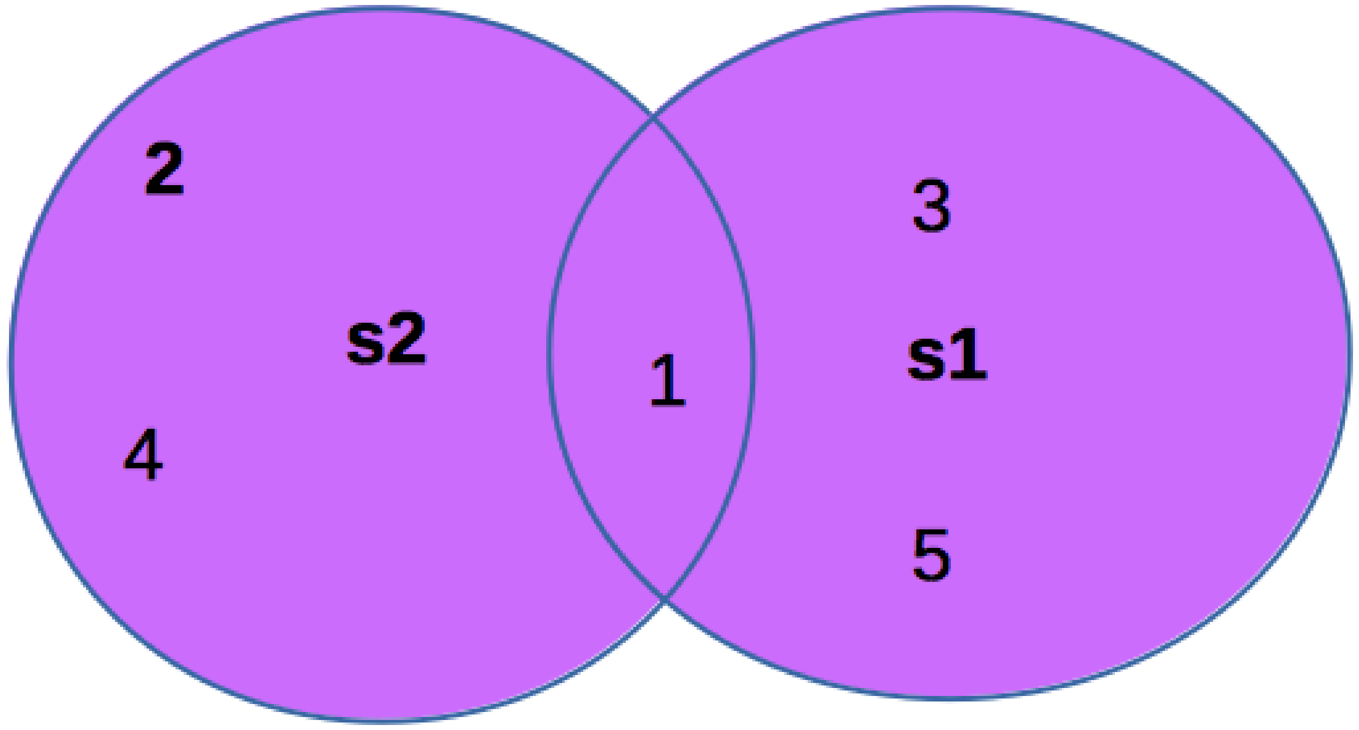


Equality Test

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

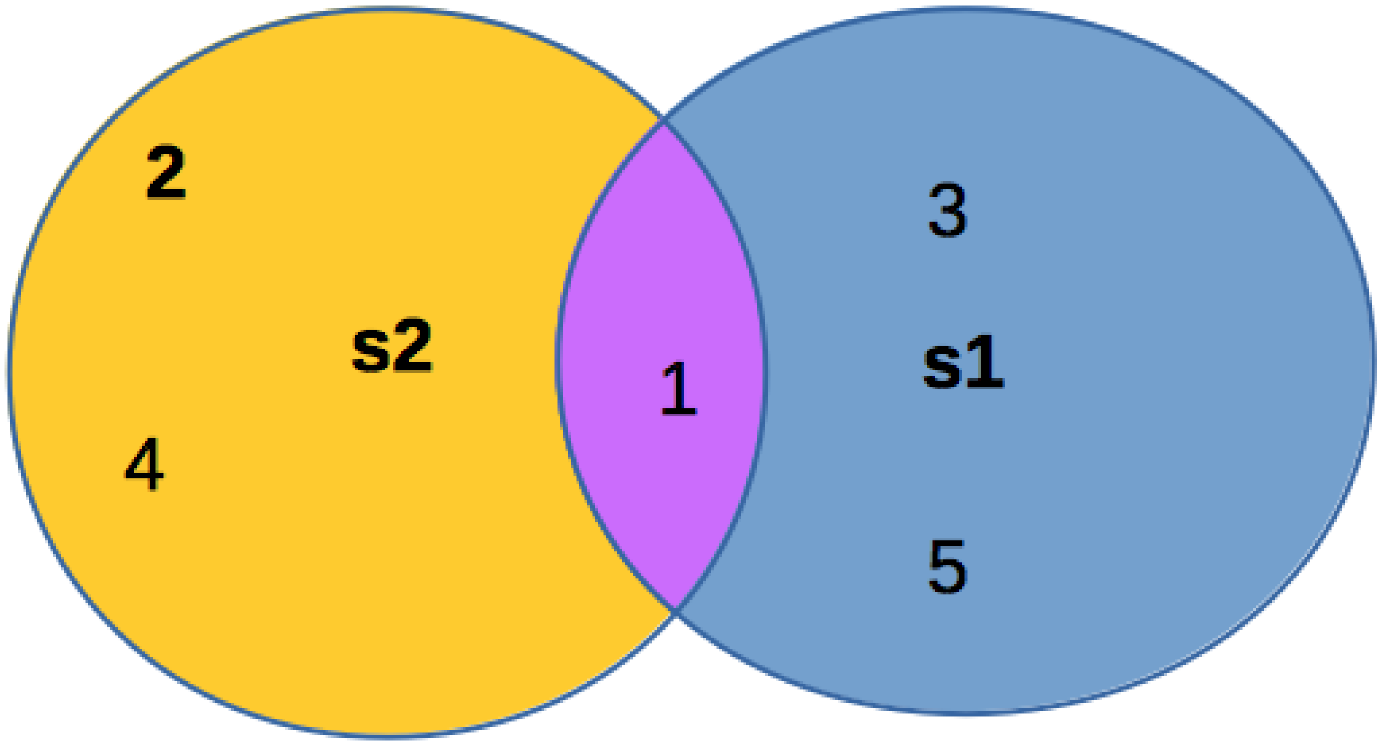
Set Operations (union, |)

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



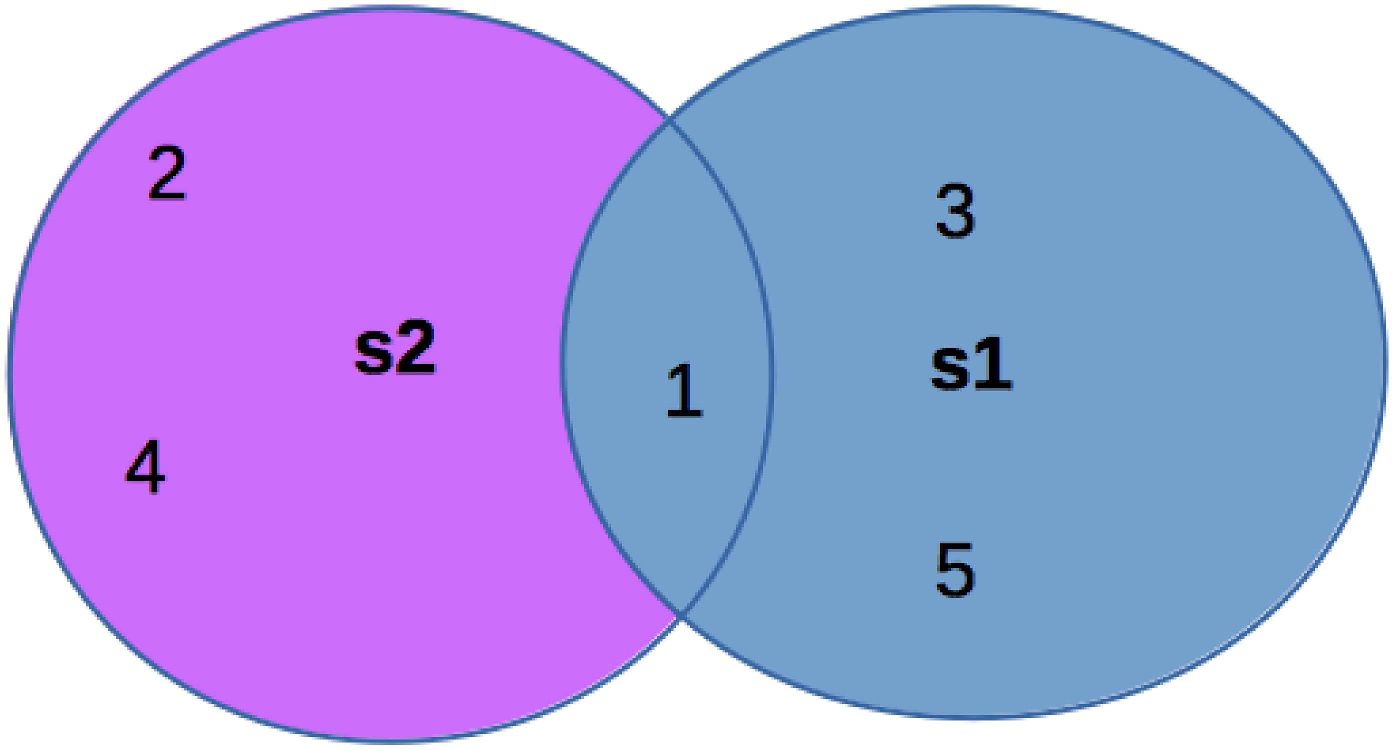
Set Operations (intersection, &)

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



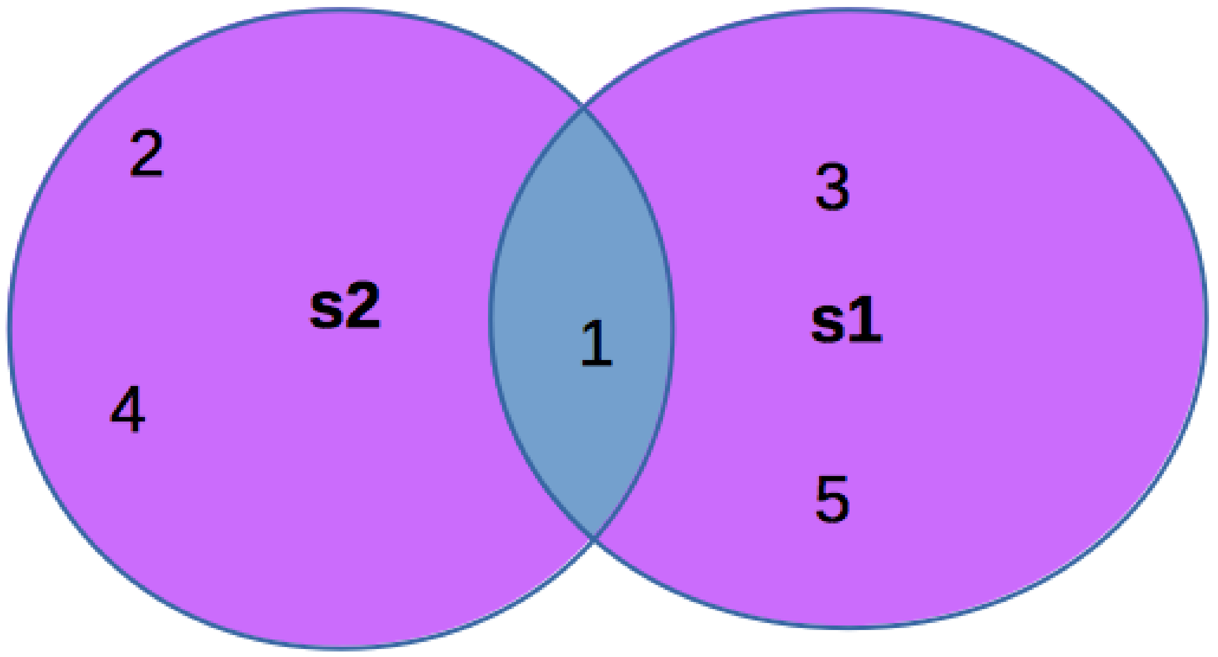
Set Operations (difference, -)

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



Set Operations (symmetric_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

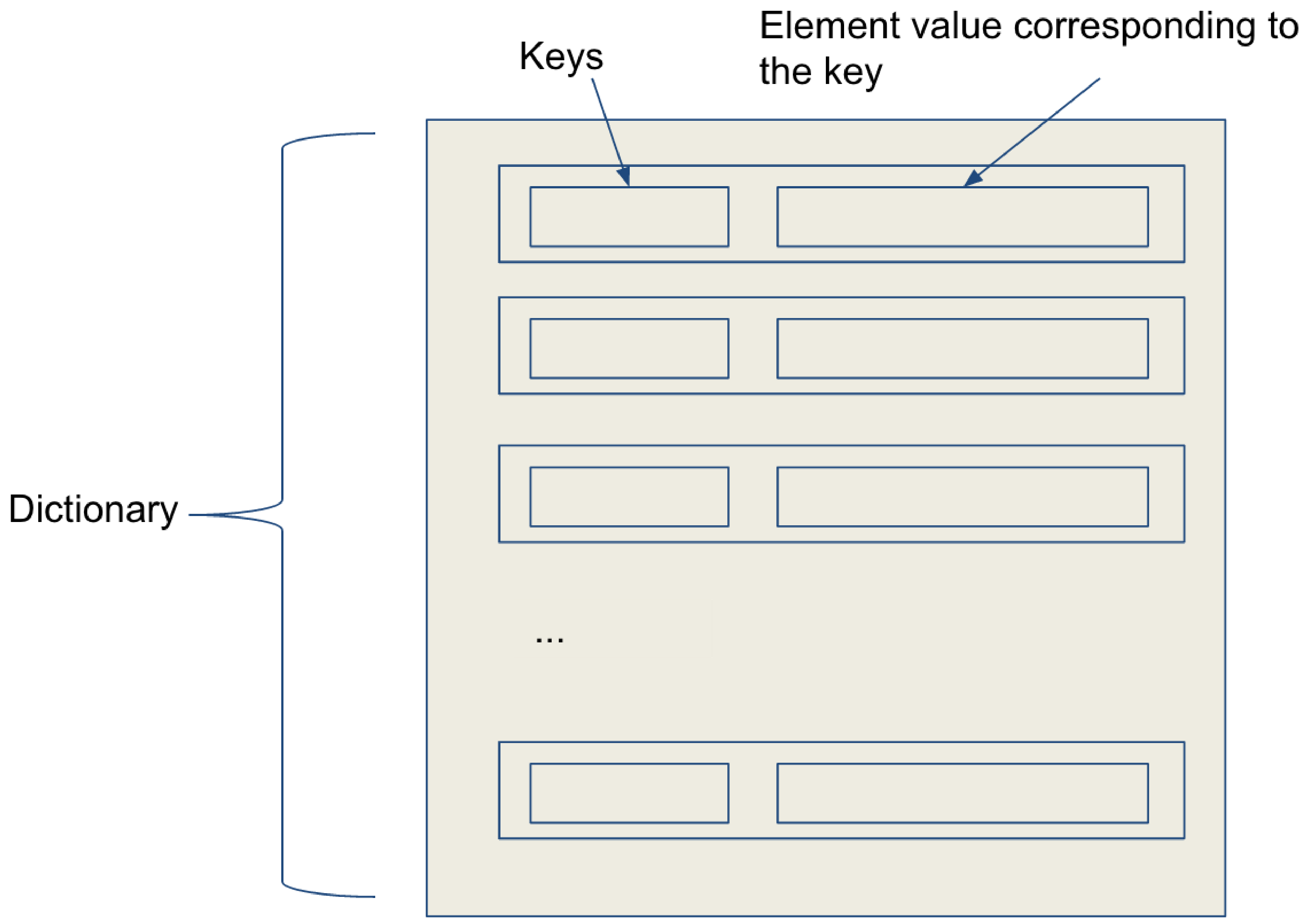
Set and List performance compared:

- using the time library: SetListPerformanceTest.py

Dictionary

- Why dictionary?
- Suppose your program stores a million students and frequently searches for a student using the student number. An efficient data structure for this task is a dictionary. A dictionary is a collection that stores keys associated with elements.
- The key works similarly to an indexer. In order to get the element associated, you need to provide the key.

Key/value pairs



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}
```

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]))
```

The len and in operators

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

in operator checks whether the key on the left exists in the dictionary on the right.

```
>>> 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.
popitem(): tuple	Returns a randomly-selected key/value pair as a tuple and removes the selected entry

Exercise: Guess the capital

- Write a program that prompts the user to enter the capital of a random country.
- Upon receiving 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

```
import random

from list_of_countries import COUNTRIES

def main():
    countries = list(COUNTRIES.keys())
    country_to_guess = random.choice(countries)
    capital = input("What is the capital of "
                    + country_to_guess + "? ").strip()

    if capital.lower() == COUNTRIES[country_to_guess]\
            .lower():
        print("Your answer is correct")
    else:
        print("The correct answer should be "
              + COUNTRIES[country_to_guess])

main()
```

Introduction to programming with Python

Session 6-2

Objectives

- To open a file, read/write data from/to a file
- To use file dialogs for opening and saving data
- To read data from a Web resource
- To handle exceptions using the try/except/finally clauses
- To raise exceptions using the raise statements
- To become familiar with Python’s built-in exception classes
- To access exception object in the handler

Open a File

We create a **file object** with the following syntax:

```
file = open(filename, mode)
```

Mode	Description
r	Open a file for reading only
r+	Open a file for both reading and writing
w	Open a file for writing only
a	Open a file for appending data. Data are written to the end of the file
rb	Open a file for reading binary data
wb	Open a file for writing binary data

Write to a File: Example

A program that creates a file if it does not exist (an existing file with the same name will be erased) and writes in it:

```
def main():
    # Open file for output
    outfile = open("Python_projects.txt", "w")
    # Write data to the file
    outfile.write("Django\n")
    outfile.write("Flask\n")
    outfile.write("Ansible")
    outfile.close() # Close the output file

main() # Call the main function
```

Test whether a file exists

```
import os.path

if os.path.isfile("Python_projects.txt"):
    print("Python_projects.txt exists")
```

Reading from a File

After a file is opened for reading data, you can use:

- the **read(size)** method to read a specified number of characters or all characters,
- the **readline()** method to read a whole line
- the **readlines()** method to read all lines into a list.

Read from a File: Example with read()

```
def main():
    # Open file for input
    infile = open("Python_projects.txt", "r")
    print("Using read(): ")
    print(infile.read())
    infile.close() # Close the input file
main() # Call the main function
```

Read from a File: Example with read(size)

```
def main():
    infile = open("Python_projects.txt", "r")
    print("\nUsing read(number): ")
    s1 = infile.read(4)  # read till the 4th character
    print(s1)
    s2 = infile.read(10)  # read from 4th till 4th+10th
    print(repr(s2))  # a new line is also a character \n
    infile.close()  # Close the input file

main()  # Call the main function
```

Read from a File: Example with readline()

```
def main():
    infile = open("Python_projects.txt", "r")
    print("\nUsing readline(): ")
    line1 = infile.readline()
    line2 = infile.readline()
    line3 = infile.readline()
    line4 = infile.readline()
    print(repr(line1))
    print(repr(line2))
    print(repr(line3))
    print(repr(line4))
    infile.close() # Close the input file

main() # Call the main function
```

Read from a File: Example with readlines()

```
def main():
    # Open file for input
    infile = open("Python_projects.txt", "r")
    print("\n(4) Using readlines(): ")
    print(infile.readlines())   # a list of lines
    infile.close()   # Close the input file

main() # Call the main function
```

Append Data to a File

You can use the 'a' mode to open a file for appending data to an existing file.

```
def main():
    # Open file for appending data
    outfile = open("Info.txt", "a")
    outfile.write("\nPython is interpreted\n")
    outfile.close() # Close the input file

main() # Call the main function
```

Writing/Reading Numeric Data

To write numbers, convert them into strings, and then use the write method to write them to a file. In order to read the numbers back correctly, you should separate the numbers with a whitespace character such as " " (empty string) or '\n' (new line).

Writing/Reading Numeric Data: Example

```
from random import randint

def main():
    # Open file for writing data
    outfile = open("Numbers.txt", "w")
    for i in range(10):
        outfile.write(str(randint(0, 9)) + " ")
    outfile.close() # Close the file
    # Open file for reading data
    infile = open("Numbers.txt", "r")
    s = infile.read()
    numbers = [int(x) for x in s.split()]
    for number in numbers:
        print(number, end = " ")
    infile.close() # Close the file

main() # Call the main function
```

Exercise

Write a program that prompts the user to enter a file and counts the number of occurrences of each letter in the file regardless of case.

Only take the characters of the alphabet, you can get them with the following

```
from string import ascii_lowercase
print(ascii_lowercase)  # abcdefghijklmnopqrstuvwxyz
```

Solution

```
from string import ascii_lowercase
from pprint import pprint

def main():
    filename = input("Enter a filename: ").strip()
    dict_of_letter = {}
    f = open(filename)
    for line in f:
        for letter in line.lower():
            if letter in ascii_lowercase:
                if letter in dict_of_letter:
                    dict_of_letter[letter] += 1
                else:
                    dict_of_letter[letter] = 1
    f.close()
    pprint(dict_of_letter)

main()
```

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 CountOccurrenceOfWords.py

Retrieving Data from the Web

Using Python, you can write simple code to read data from a website. All you need to do is to open a URL link using the urlopen function as follows:

```
import urllib.request
infile = urllib.request.urlopen('http://city.ac.uk/')
html_page = infile.read().decode()
print(html_page)
```

It represents the full HTML of the page just as a web browser would see it

Exercise

- Count each letter from a web page (from the source code of the page)
- You can reuse the code of the previous and try to refactor so that both programs use the same count_letter function

...ution

```
from pprint import pprint
from string import ascii_lowercase
import urllib.request

def main():
    url = input("Enter an URL for a file: ").strip()
    infile = urllib.request.urlopen(url)
    f = infile.read().decode() # Read the content as str
    dict_of_letter = {}
    for line in f:
        for letter in line.lower():
            if letter in ascii_lowercase:
                if letter in dict_of_letter:
                    dict_of_letter[letter] += 1
                else:
                    dict_of_letter[letter] = 1
    pprint(dict_of_letter)

main()
```

Exception Handling

What happens if the user enters a file or an URL that does not exist? The program would raise an error and abort. For example, if you run count_letter.py with an incorrect input:

```
u:\python1\session6\python count_letter.py
Enter a filename: non_existant_file.txt
Traceback (most recent call last):
  File "count_letter.py", line 18, in  <module>
    main()
  File "count_letter.py", line 7, in main
    f = open(filename)
FileNotFoundError: [Errno 2]
No such file or directory: 'non_existant_file.txt'

Process finished with exit code 1
```

The try … except clause

Catching one exception type

```
try:
  <body>
except <ExceptionType>:
  <handler>
```

The try ... except clause

Catching several exception types

```
try:
 <body>
except <ExceptionType>:
 <handler1>
 <handler1>
...
except <ExceptionTypeN>:
 <handlerN>
except:
 <handlerExcept>
else:
<process_else> # will be executed if not exception
finally:
<process_finally> # executed with or without exception
```

Example

```
def main():
    try:
        number1, number2 = int(
          input("Enter two integers,"
                "separated by a comma: "))
        result = number1 / number2
        print("Result is " + str(result))
    except ZeroDivisionError:
        print("Division by zero!")
    except SyntaxError:
        print("A comma may be missing in the input")
    except:
        print("Something wrong in the input")
    else:
        print("No exceptions")
    finally:
        print("The finally clause is executed")
main()
```

Raising Exceptions

You learned how to write the code to handle exceptions in the previous section. Where does an exception come from? How is an exception created? Exceptions are objects and objects are created from classes. An exception is raised from a function. When a function detects an error, it can create an object of an appropriate exception class and raise the object, using the following syntax:

```
raise ExceptionClass("Something is wrong")
```

Processing Exceptions Using Exception Objects

You can access the exception object in the except clause with the **as** keyword.

```
try:
    number = int(input("Enter a number: "))
    print("The number entered is", number)
except NameError as ex:
    print("Exception:", ex)
```

Using the with statement

It is good practice to use the **with** keyword when dealing with file objects. This has the advantage that the file is properly closed after its suite finishes, even if an exception is raised on the way. It is also much shorter than writing equivalent try-finally blocks:

```
with open('Python_projects.txt', 'r') as f:
    read_data = f.read()

assert f.closed
```

The json file format

- json (JavaScript Object Notation) is a lightweight **data interchange format** with which you:
  - dump data ("serialize")
  - load data ("deserialize")

```
import json
serialized_data = json.dumps(
    ['foo', {'bar': ('baz', None, 1.0, 2)}])
print(serialized_data)
deserialized_data = json.loads(serialized_data)
print(deserialized_data)
```

Example with a simple REST API (1)

How to get the capital of each country?

```
import json
from urllib import request

infile = request.urlopen(
    'https://restcountries.eu/rest/v1/all')
content_as_python_obj = json.loads(infile.read().decode(
for country in content_as_python_obj:
    print(country['borders'])
```

Can you see what object is the "borders"?

Example with a simple REST API (2)

```
import json
from urllib import request

infile = request.urlopen(
    'https://restcountries.eu/rest/v1/all')
content_as_python_obj = json.loads(infile.read().decode(
for country in content_as_python_obj:
    print(country['capital'])
```

API

- In the previous case, an API (Application Programming Interface) is simply a specification of remote calls exposed to the API consumers
- We are using the API as a service by just calling (doing a GET) its available urls

Example with the Twitter API using the client Tweepy

1. Navigate to https://apps.twitter.com/
2. Click the button to create a new application
3. Enter dummy data
4. Once the application is created, get the following:
    - consumer_key
    - consumer_secret
    - access_token
    - access_secret

Get tweet with #python

```
import tweepy

consumer_key = 'get_your_own'
consumer_secret = 'get_your_own'
access_token = 'get_your_own'
access_secret = 'get_your_own'

def main():
    auth = tweepy.auth.OAuthHandler(consumer_key,
                                    consumer_secret)
    auth.set_access_token(access_token, access_secret)
    api = tweepy.API(auth)

    tweets = api.search(q='#python')
    for t in tweets:
        print(t.created_at, t.text, '\n')
main()
```