Unity
Thống nhất
Excellence
Vượt trội
Leadership
Tiên phong

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KINH TẾ - LUẬT

Chapter 4. WORKING WITH FILES

Course: Programming

www.uel.edu.vn

OBJECTIVES

After completing this chapter, you should be able to

- Select text files and binary files to read and write data.
- Demonstrate the use of built-in functions to navigate the file system.
- Make use of os module to operate on underlying Operating System tasks.

CONTENTS

- INTRODUCTION OF DIFFERENT FILE TYPES IN PYTHON
- INTERACT WITH FILES WITH THE HELP OF BUILT-IN FUNCTION
- PERFORM OPERATIONS SUCH AS READ, WRITE, AND MANIPULATE FILES

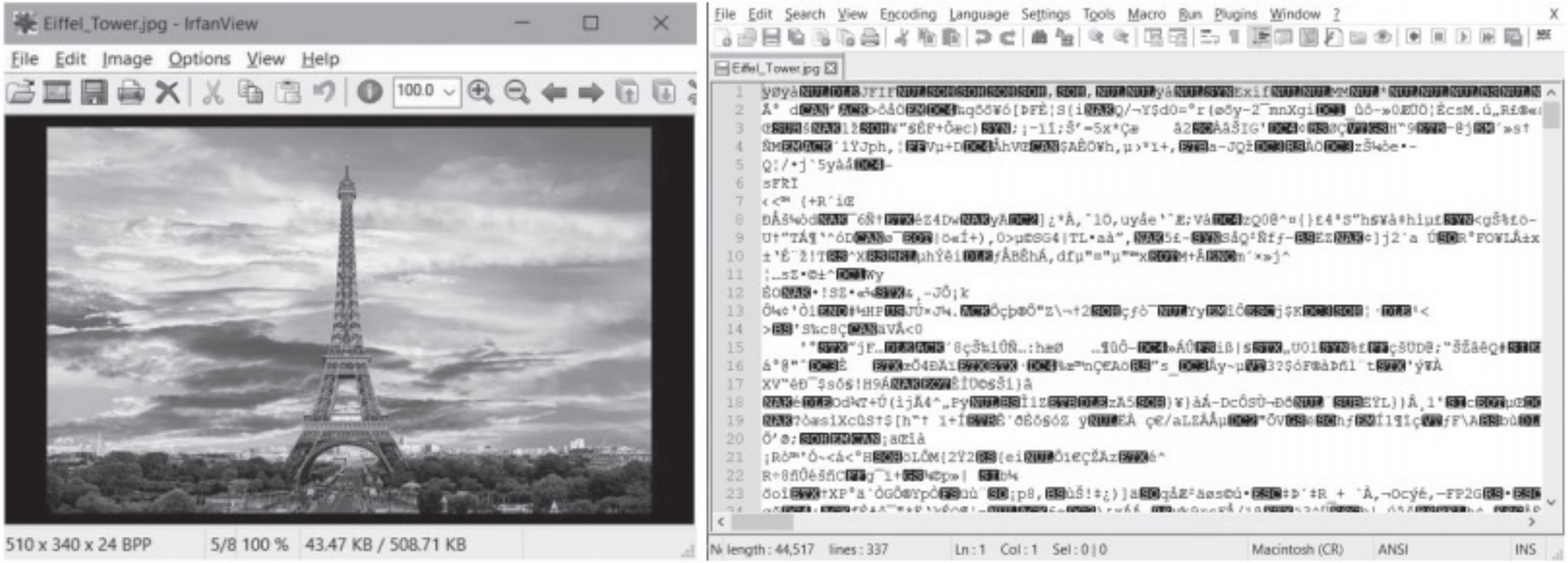
TYPES OF FILES

IMAP
.EML
.log
JPG
.msg
MPEG4
.rar
.MSM
.html
.HTTP
PDF
Adobe
.PST
GIF
WAV
MP3
.wma
.ini
.POP3
.OST
M4A
.iso
AVI

TYPES OF FILES

Binary
Text

TYPES OF FILES

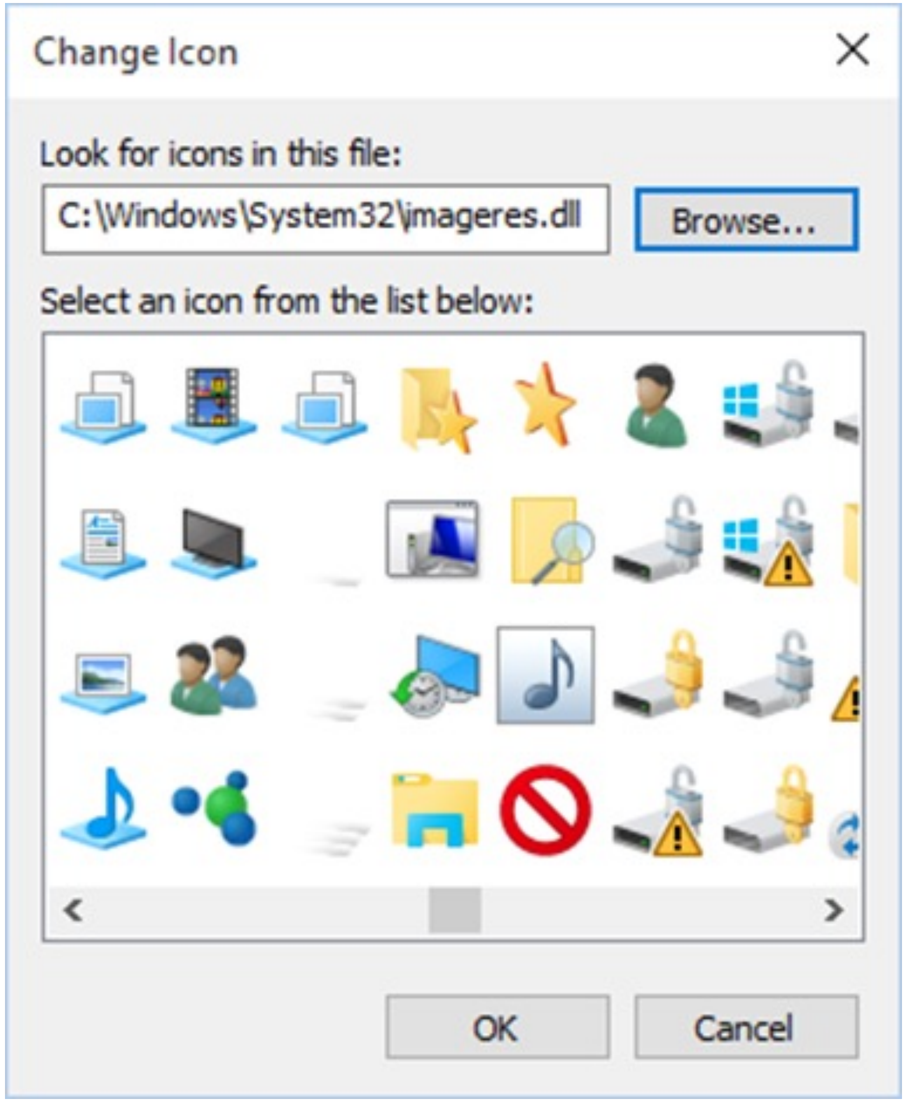

Eiffel_Tower.jpg - IrfanView
File Edit Image Options View Help
100.0
510 x 340 x 24 BPP
5/8 100 %
43.47 KB / 508.71 KB
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Eiffel_Tower.jpg
length : 44,517 lines : 337
Ln : 1 Col : 1 Sel : 0 | 0
Macintosh (CR)
ANSI
INS

Images and its binary content

TYPES OF FILES

Text files

FILE PATHS



Look for icons in this file: C:\Windows\System32\imageres.dll

In the Windows Operating System, the maximum length for a path is 260 characters and in the Linux Operating System the maximum path length is of 4096 characters.

FULLY QUALIFIED VS RELATIVE FILE PATH

The fully qualified path (Absolute path): A path points to the file location, which always contains the root and the complete directory list

Ex:

- *"C:\python.txt"*
- *"C:\learning\python.txt"*

FULLY QUALIFIED VS RELATIVE FILE PATH

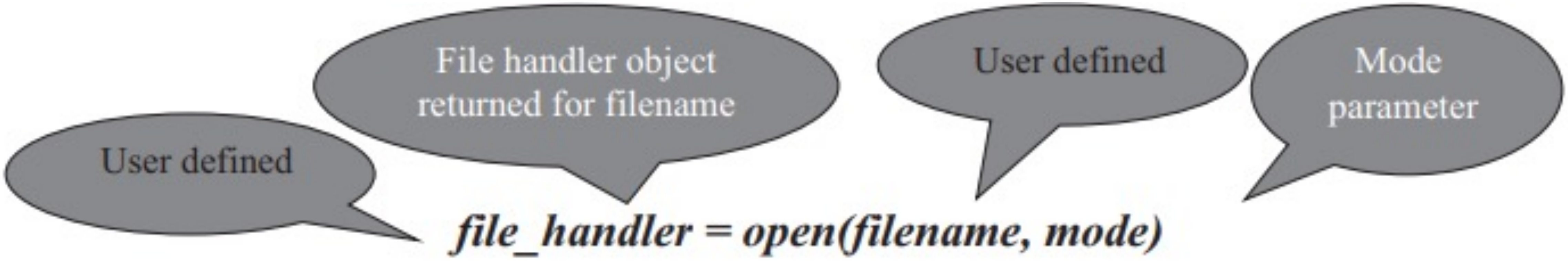
The relative path: A path contains “double-dots”; that is, two consecutive periods together as one of the directory components in a path or “single-dot” that is, one period as one of the directory components in a path.

Ex:

- *"..\python.txt"*
- *".\python.txt"*

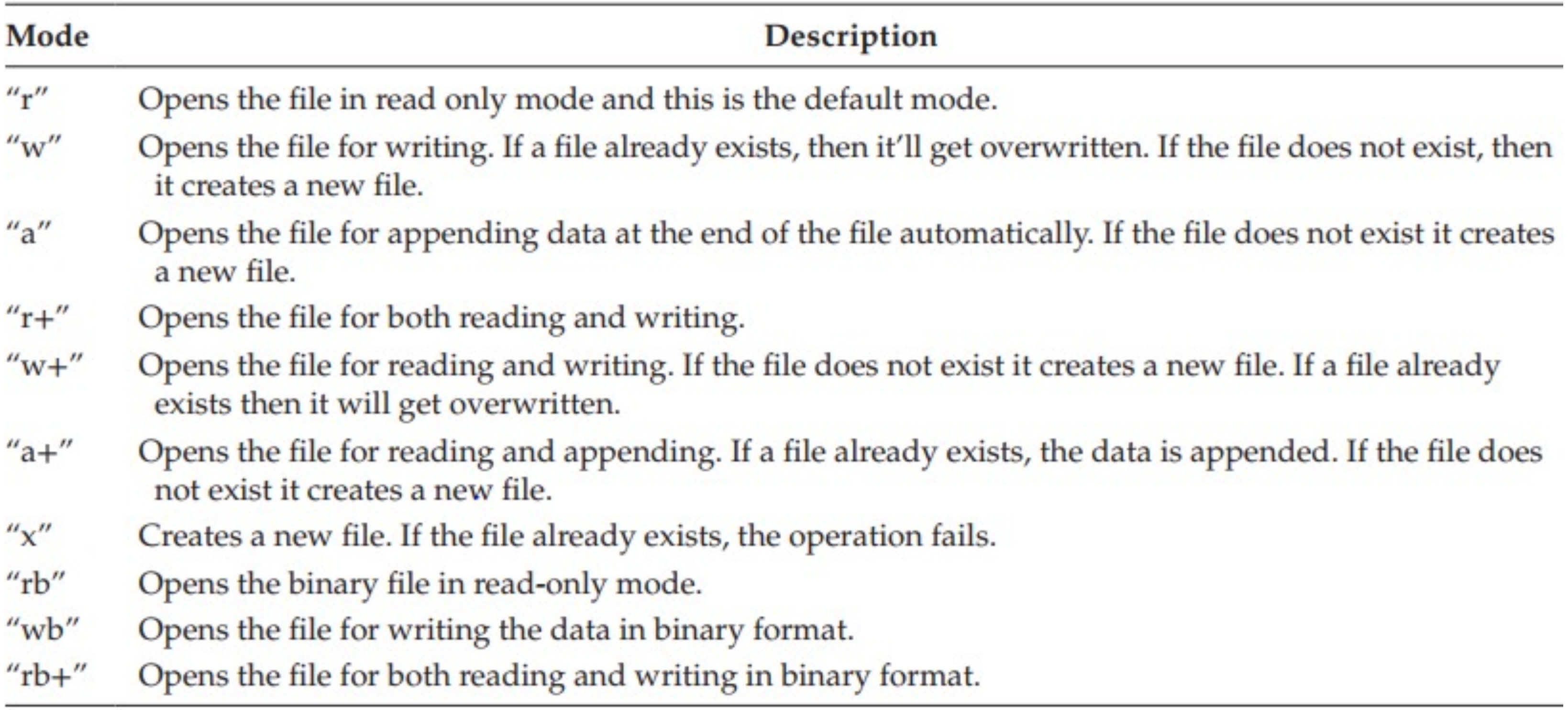
OPERATIONS WITH FILES

1. File open() method


File handler object returned for filename
User defined
User defined
Mode parameter
file_handler = open(filename, mode)

OPERATIONS WITH FILES

1. File open() method



Mode	Description
“r”	Opens the file in read only mode and this is the default mode.
“w”	Opens the file for writing. If a file already exists, then it’ll get overwritten. If the file does not exist, then it creates a new file.
“a”	Opens the file for appending data at the end of the file automatically. If the file does not exist it creates a new file.
“r+”	Opens the file for both reading and writing.
“w+”	Opens the file for reading and writing. If the file does not exist it creates a new file. If a file already exists then it will get overwritten.
“a+”	Opens the file for reading and appending. If a file already exists, the data is appended. If the file does not exist it creates a new file.
“x”	Creates a new file. If the file already exists, the operation fails.
“rb”	Opens the binary file in read-only mode.
“wb”	Opens the file for writing the data in binary format.
“rb+”	Opens the file for both reading and writing in binary format.

Access modes of the files

OPERATIONS WITH FILES

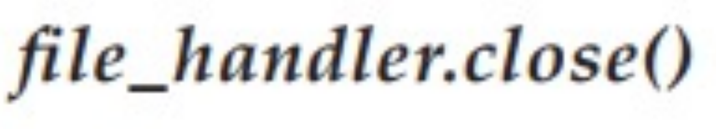
2. File open() method

Example:

```
>>> file_handler = open("example.txt","x")
>>> file_handler = open(“example.txt","r")
>>> file_handler = open("C:\example.txt","r")
>>> file_handler = open("C:\learning\example.txt","r")
>>> file_handler = open("C:\\forme\\byme.txt","r")
>>> file_handler = open("C:\\noway\python.txt","r")
>>> file_handler = open(r"C:\noway\python.txt","r")
>>> file_handler = open(“example.txt","r")
```

OPERATIONS WITH FILES

2. File close() method



file_handler.close()

Example:

```
>>> file_handler = open(“python.txt","r")

>>> file_handler.close()
```

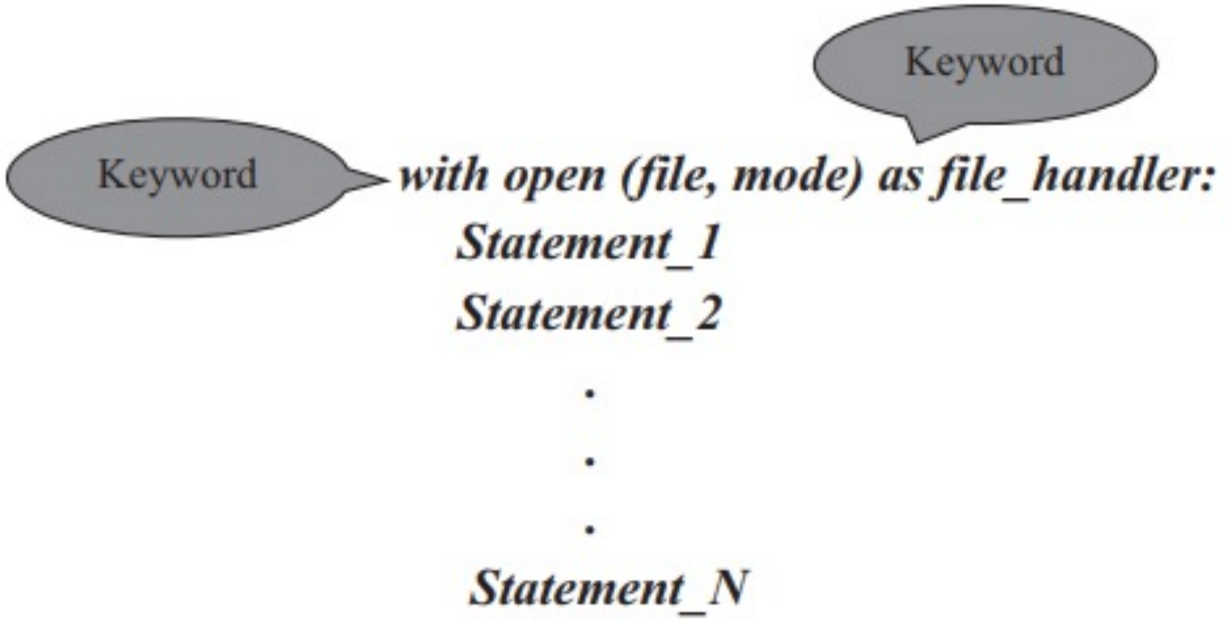
OPERATIONS WITH FILES

2. File close() method

```
try:
    f = open("file", "w")
    try:
        f.write(Write something here!')
    finally:
        f.close()
except IOError:
    print(‘File error!')
```

OPERATIONS WITH FILES

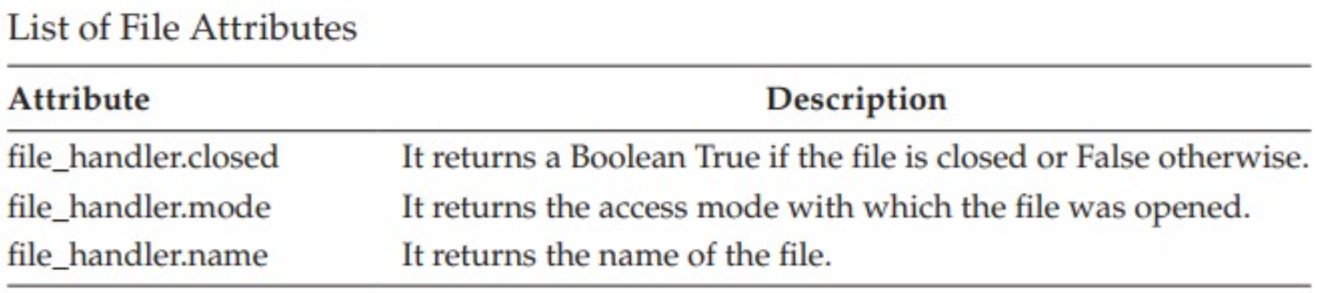
2. File close() method



```
with open (file, mode) as file_handler:
    Statement_1
    Statement_2
    .
    .
    .
    Statement_N
```

OPERATIONS WITH FILES

3. File object attributes



List of File Attributes

Attribute	Description
file_handler.closed	It returns a Boolean True if the file is closed or False otherwise.
file_handler.mode	It returns the access mode with which the file was opened.
file_handler.name	It returns the name of the file.

Example:

```
>>> file_handler = open(“python.txt", "w")

>>> print(f"File Name is {file_handler.name}")

>>> print(f"File State is {file_handler.closed}")

>>> print(f"File Opening Mode is {file_handler.mode}")
```

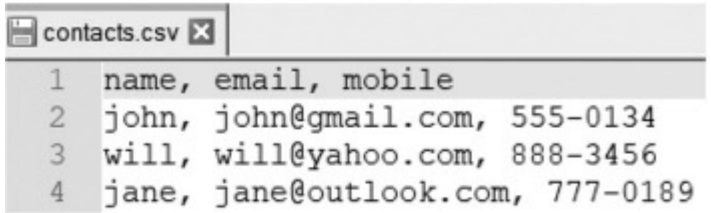
OPERATIONS WITH FILES

4. Read and write data

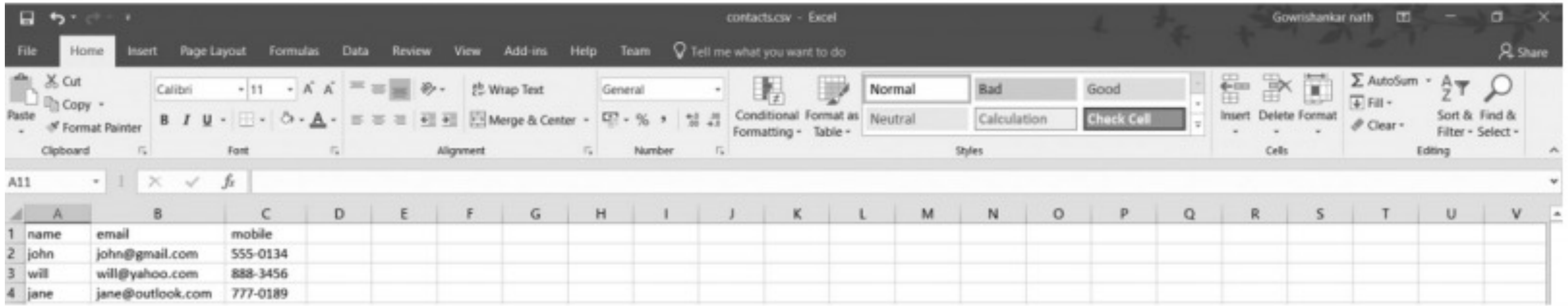
Method	Syntax
read()	file_handler.read([size])
readline()	file_handler.readline()
readlines()	file_handler.readlines()
write()	file_handler.write(string)
writeline()	file_handler.writelines(sequence)
tell()	file_handler.tell()
seek()	file_handler.seek(offset,from_what)

CSV FILES

Read and write CSV (Comma Separated Values) files



```
name, email, mobile
john, john@gmail.com, 555-0134
will, will@yahoo.com, 888-3456
jane, jane@outlook.com, 777-0189
```



	A	B	C
1	name	email	mobile
2	john	john@gmail.com	555-0134
3	will	will@yahoo.com	888-3456
4	jane	jane@outlook.com	777-0189

Using built-in function module called csv, some methods are:

>>>csv.reader(csvfile)

>>>csv.writer(csvfile)

>>>csvwriter.writerow(row)

>>>csvwriter.writerows(rows)

CSV FILES

Read and write CSV (Comma Separated Values) files

Example

```
import csv
def main():
    with open(‘example.csv', newline='') as csvfile:
    csv_reader = csv.reader(csvfile)
    print("Print each row in CSV file")
    for each_row in csv_reader:
    print(",".join(each_row))
if __name__ == "__main__":
    main()
```

EXCEL FILES

Using openpyxl

Openpyxl is a Python library that provides various methods to interact with Excel Files using Python. It allows operations like reading, writing, arithmetic operations, plotting graphs, etc.

```
from openpyxl import Workbook

workbook = Workbook()

sheet = workbook.active

sheet["A1"] = "hello"

sheet["B1"] = "world!"

workbook.save(filename="hello_world.xlsx")
```

EXCEL FILES

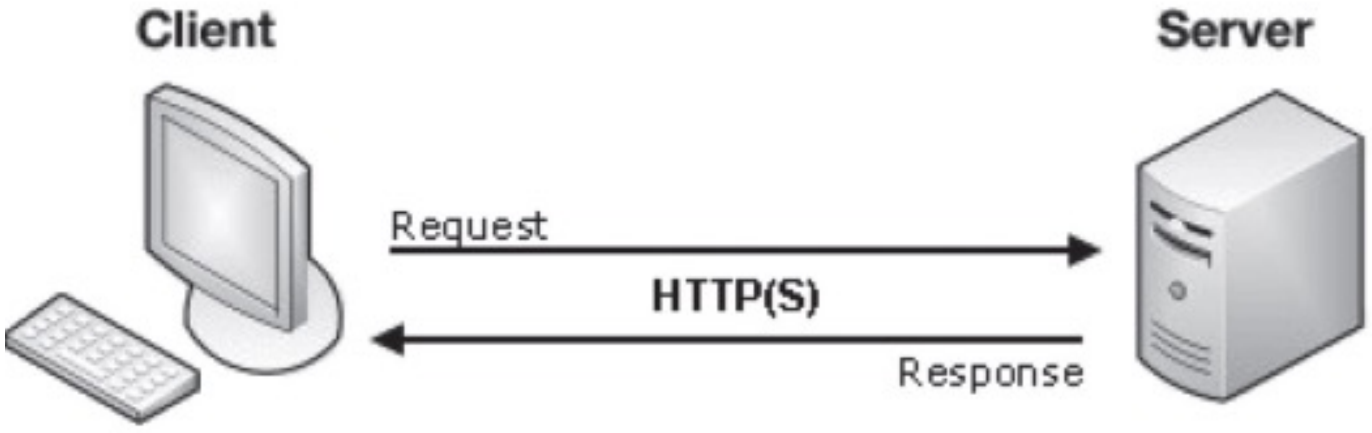
Using pandas

pandas is a Python library that provides fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

```
>>>import panda as pd
>>>df = pd.DataFrame({'a':pd.Series([1, 2]), 'b':pd.Series([10, 20, 30, 40, 50]), 'c':pd.Series([100, 200, 300])})
>>> df_csv = pd.read_csv("foo.csv")
>>> df _excel = pd.read_excel("foo.xlsx")
>>> df.to_csv('foo.csv')
>>> df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

JSON AND XML FILES

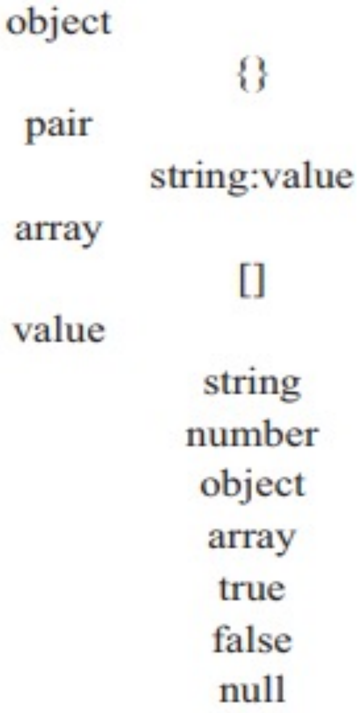
JSON (JavaScript Object Notation) and XML (EXtensible Markup Language) standards are commonly used for transmitting data in web applications.



JSON FILES

The built-in data types in JSON are strings, numbers, booleans (i.e., true and false), null, objects, and arrays. JSON is built on two structures:

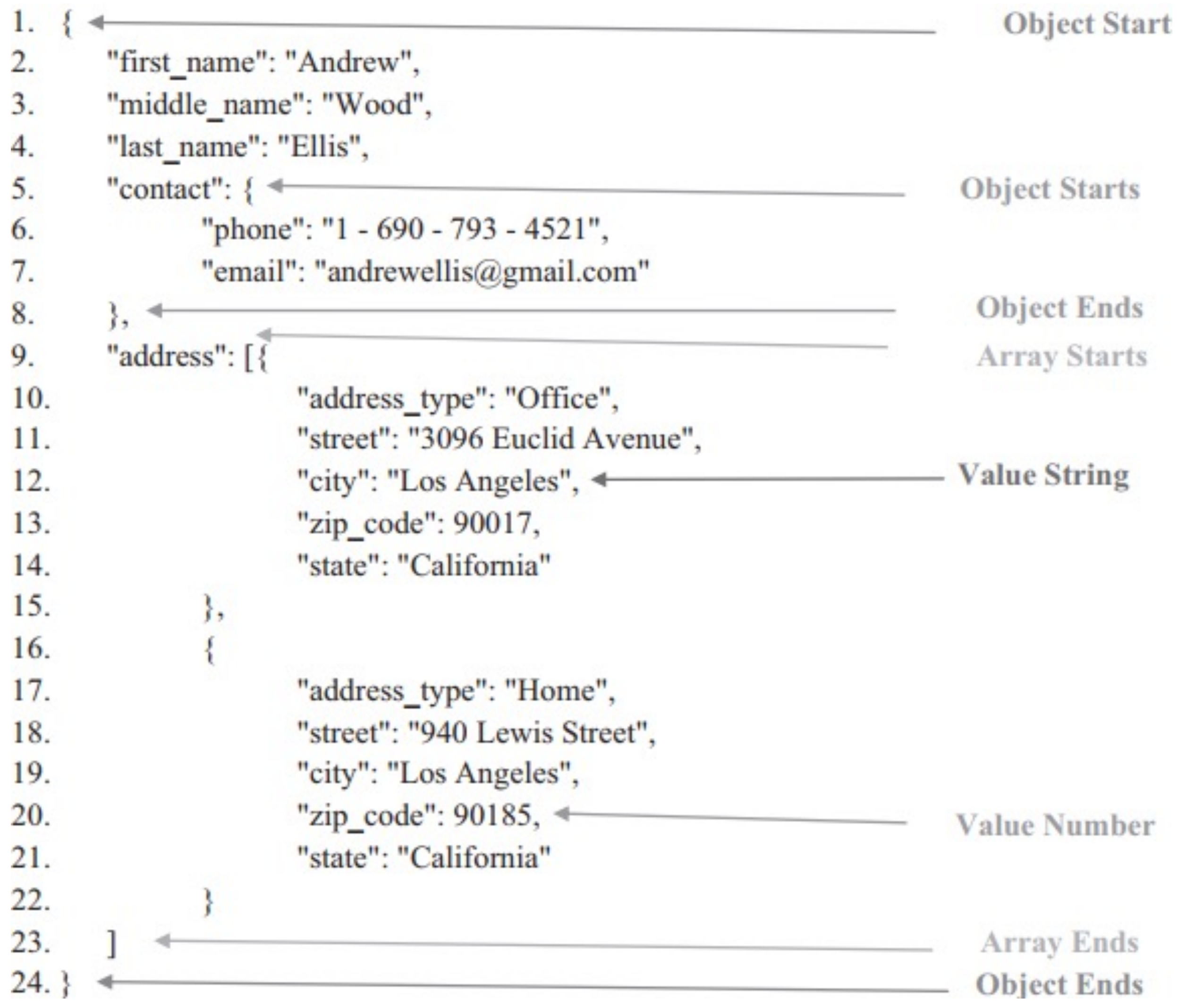
- A collection of *string: value* properties. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.



object	{}
pair	string:value
array	[]
value	string number object array true false null

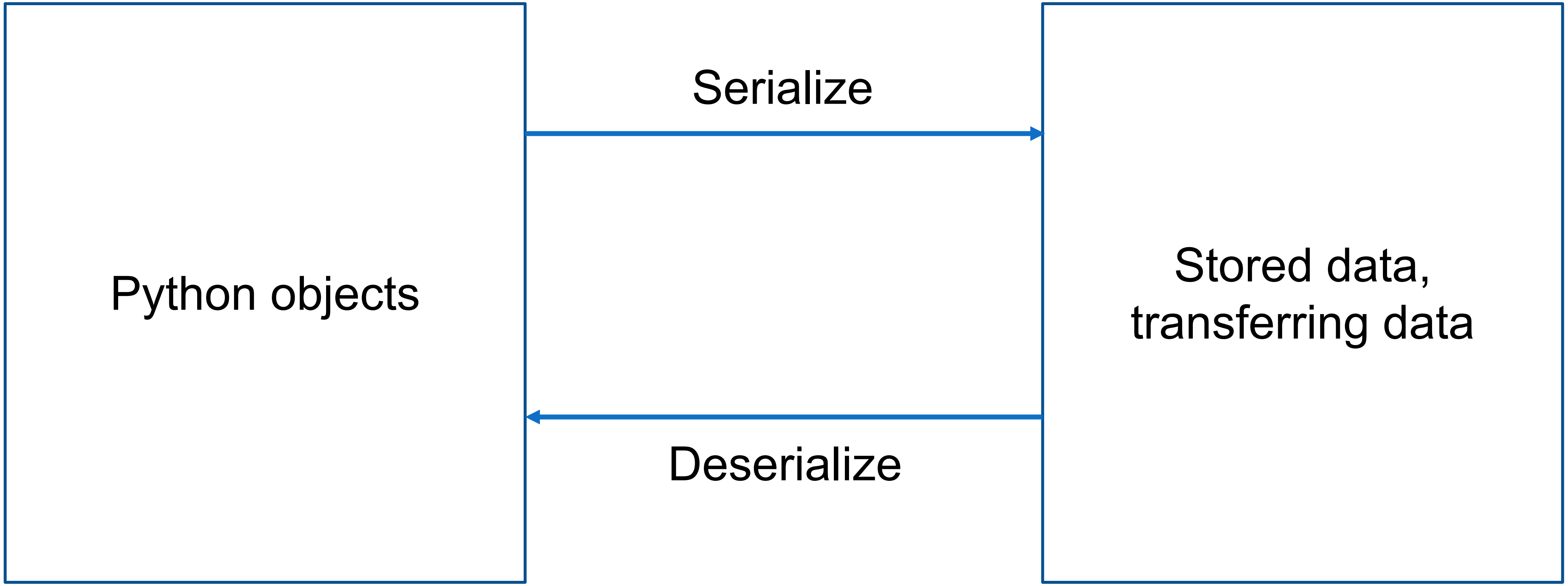
JSON Structures

JSON FILES



```
{
    "first_name": "Andrew",
    "middle_name": "Wood",
    "last_name": "Ellis",
    "contact": {
            "phone": "1 - 690 - 793 - 4521",
            "email": "andrewellis@gmail.com"
    },
    "address": [{
                    "address_type": "Office",
                    "street": "3096 Euclid Avenue",
                    "city": "Los Angeles",
                    "zip_code": 90017,
                    "state": "California"
            },
            {
                    "address_type": "Home",
                    "street": "940 Lewis Street",
                    "city": "Los Angeles",
                    "zip_code": 90185,
                    "state": "California"
            }
    ]
}
```

JSON FILES



Python objects

Serialize

Deserialize

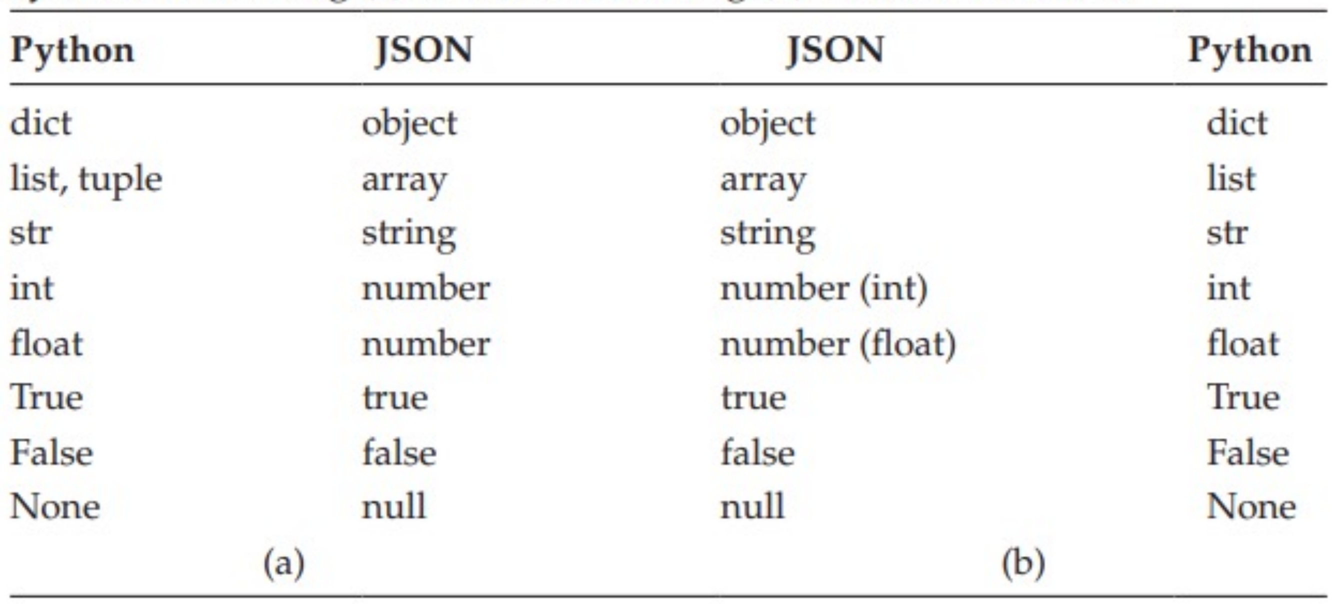
Stored data, transferring data

JSON FILES

The Python **json** module provides methods **load()** for turning JSON encoded data into Python objects from a file and **loads()** methods for turning JSON encoded data into Python objects from a string.

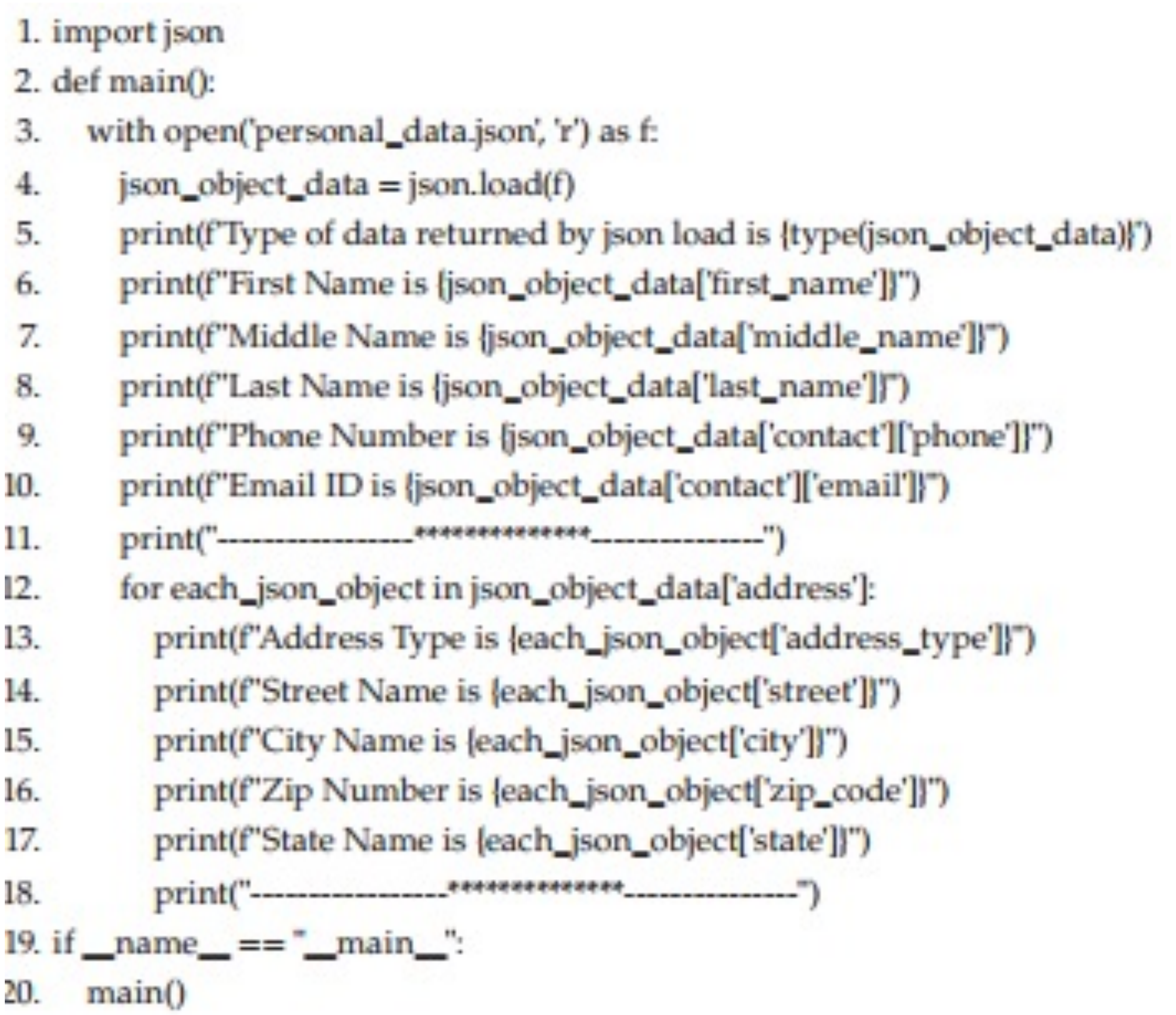
The Python **json** module provides methods **dump()** for writing data to JSON file and **dumps()** for writing to a Python string

Python Serializing (a) and Deserializing (b) Conversion Table



Python	JSON	JSON	Python
dict	object	object	dict
list, tuple	array	array	list
str	string	string	str
int	number	number (int)	int
float	number	number (float)	float
True	true	true	True
False	false	false	False
None	null	null	None
(a)		(b)	

JSON FILES



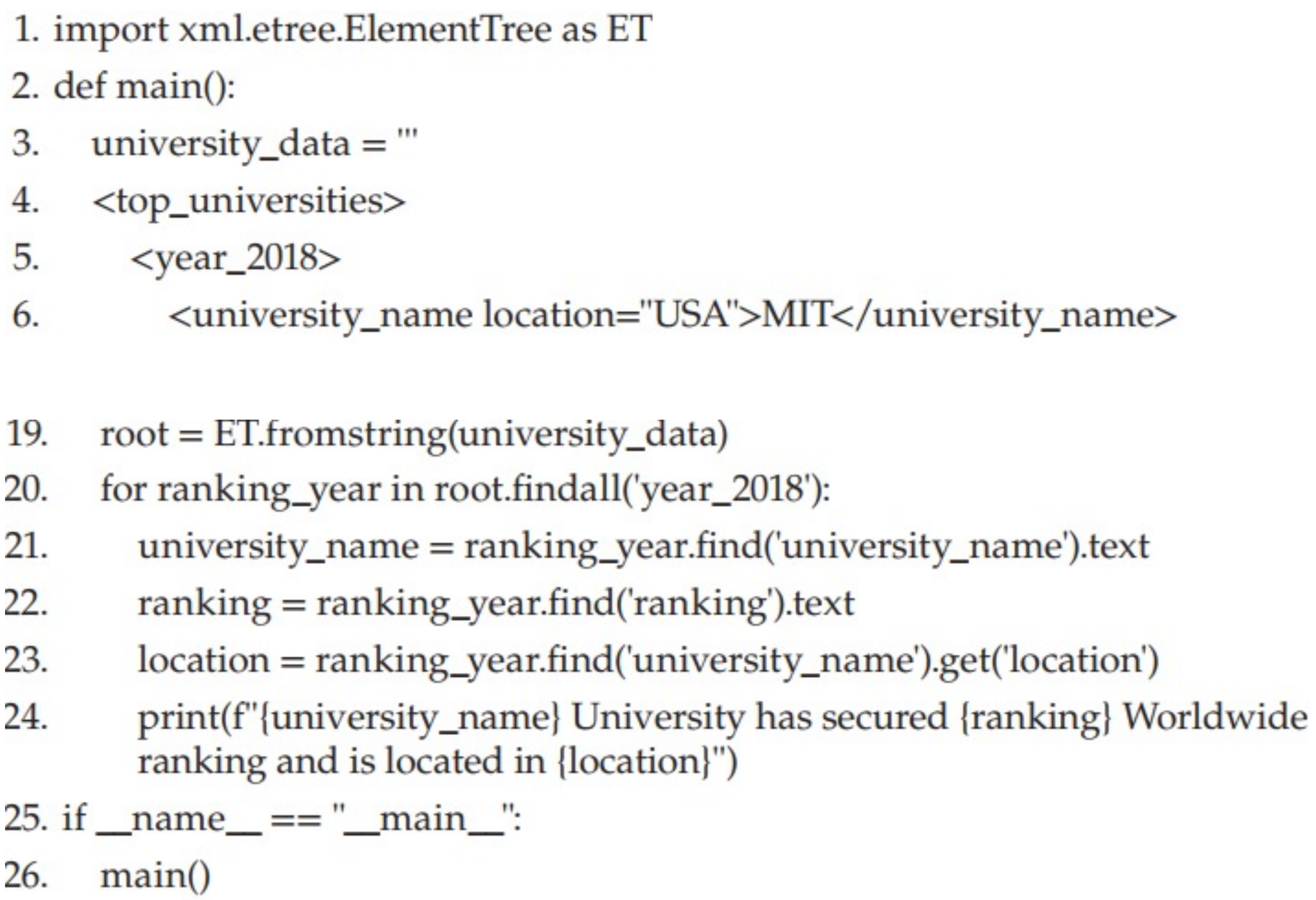
```
import json
def main():
    with open('personal_data.json', 'r') as f:
        json_object_data = json.load(f)
        print(f'Type of data returned by json load is {type(json_object_data)}')
        print(f"First Name is {json_object_data['first_name']}")
        print(f"Middle Name is {json_object_data['middle_name']}")
        print(f"Last Name is {json_object_data['last_name']}")
        print(f"Phone Number is {json_object_data['contact']['phone']}")
        print(f"Email ID is {json_object_data['contact']['email']}")
        print("----------------**************--------------")
        for each_json_object in json_object_data['address']:
            print(f"Address Type is {each_json_object['address_type']}")
            print(f"Street Name is {each_json_object['street']}")
            print(f"City Name is {each_json_object['city']}")
            print(f"Zip Number is {each_json_object['zip_code']}")
            print(f"State Name is {each_json_object['state']}")
            print("----------------**************--------------")
if __name__ == "__main__":
    main()
```

XML FILES

- EXtensible Markup Language (XML) document is a simple and flexible text format that is used to exchange wide variety of data on the Web and elsewhere.
- An XML document is a universal format for data on the Web. XML allows developers to easily describe and deliver rich, structured data from any application in a standard, consistent way.
- XML documents have an .xml extension.

```
<root>
    <child>
        <subchild>.....</subchild>
    </child>
</root>
```

XML FILES



```
import xml.etree.ElementTree as ET
def main():
    university_data = '''
    <top_universities>
        <year_2018>
            <university_name location="USA">MIT</university_name>

    root = ET.fromstring(university_data)
    for ranking_year in root.findall('year_2018'):
        university_name = ranking_year.find('university_name').text
        ranking = ranking_year.find('ranking').text
        location = ranking_year.find('university_name').get('location')
        print(f"{university_name} University has secured {ranking} Worldwide
        ranking and is located in {location}")
if __name__ == "__main__":
    main()
```

Construct an XML Formatted Data and Write Python Program to Parse that XML Data

XML FILES

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces

Simple API for XML (SAX):

- *This is useful when your documents are large or you have a memory limitation*
- *SAX is read-only*

XML FILES

Parsing XML with SAX API:

- SAX is a standard interface for event-driven XML parsing
- Requires you to create your own ContentHandler by subclassing xml.sax.ContentHandler
- The methods startDocument and endDocument are called at the start and the end of the XML file
- The ContentHandler is called at the start and end of each element.
- **The parse Method:**

xml.sax.parse(xmlfile, contenthandler[, errorhandler])

in which: xmlfile: This is the name of the XML file to read from

contenthandler: This must be a ContentHandler object

errorhandler: If specified, errorhandler must be a SAX ErrorHandler object.

XML FILES

Parsing XML with DOM APIs

The Document Object Model, or "DOM," is a cross-language API from the World Wide Web Consortium(W3C) for accessing and modifying XML documents

The DOM is extremely useful for random access applications

Easiest way to quickly load an XML document and to create a minidom object using the xml

XML FILES

```
from lxml import etree
from StringIO import StringIO
def parseXML(xmlFile):
f = open(xmlFile) xml = f.read()
f.close() tree= etree.parse(StringIO(xml))
print tree.docinfo.doctype
c=etree.iterparse(StringIO(xml)'''
c = etree.iterparse(xmlFile) # other method
for action,entry in c:
        text = entry.text
        print entry.tag + " => " + text
if __name__ == "__main__":
        parseXML('movie.xml')
```

IMAGE FILES

```
def main():
    with open("rose.jpg", "rb") as existing_image, open("new_rose.jpg", "wb")
    as new_image:
        for each_line_bytes in existing_image:
            new_image.write(each_line_bytes)
if __name__ == "__main__":
    main()
```

Example of creating a new image from an existing image

THANK YOU

028 37244555 www.uel.edu.vn

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KINH TẾ - LUẬT
Số 669, đường Quốc lộ 1, khu phố 3, phường Linh Xuân,
quận Thủ Đức, Thành phố Hồ Chí Minh.

UNITY - EXCELLENCE - LEADERSHIP