Programming

with python™

By
Rami Tailakh
Senior Software Engineer and Data Science Practitioner
MSc in Applied Computing and Information Technology

Day-5 Agenda

- Day-4 Quick Review
- Text File Handling
- Manipulating XML
- Manipulating JSON

Day-4 Challenges-Review

1. Write a function that takes a list of numbers and returns a list of even numbers only. The function should be one line of Python code.
2. Write a module that implements the Caesar cipher.
3. Write a function that extracts special characters from a text.
4. Write a Python code that finds the most (3) frequent words in a text.
5. Fibonacci Sequence. Again! But do not use a recursive function.

File Handling in Python

- A file stores information and has a name and an extension
- Files are stored on disks
- For efficient processing we use memory but it is volatile
- In Python we can :
  - Read a file
  - Write a file
  - Open a file, and
  - Close a file

Open a File

- We use the python **function**: open
- It is required to be assigned to a variable

```
>>> my_file =  open('file_1.txt', 'r')
>>> my_file
```

- The file should be in the same directory, or we should give its full path. Otherwise an exception will be raised.

Python Open File-Continue

- We can also use this:

```
>>> my_file =  open('file_1.txt', mode='r',encoding='utf-8')
```

- When a file **doesn't exist, the interpreter raises:**

```
FileNotFoundError: [Errno 2] No such file or directory: 'file_222.txt'
```

File Modes

Mode	Description
r	Open for read a file (default)
w	Open for writing a file
x	Open for creation exclusively, failing if the file already exists
a	Open for writing, appending to the end of the file if it exists
t	Text mode (default)
b	Binary mode
+	To open a file for updating (reading or writing)

Close a File

- It is important to close a file when it is done.
- This is important to free resources.
- We use the **method**: close

```
>>> my_file.close()
```

- With the **with** statement, it is not required to use it. The close function will be called automatically.

Read a File

- We call the **method**: read

```
>>> my_file =  open('file_1.txt', 'r')
>>> my_file.read()
```

- The file should be in the same directory, or we should give its full path
- The **with** statement can also be used:

```
>>> with open('file_1.txt','r') as my_file:
        file_text = my_file.read()
>>> print(file_text)
```

Reading (a) line(s)

- The methods **readline**() and **readlines**() are used
- The method **readline()** reads one line at a time

```
>>> my_file =  open('file_1.txt', 'r')
>>> my_file.readline()
```

- The **readlines()** method reads all lines of the whole file and saves them as items in a **list**

Iteration over lines of a File

- You can iterate through lines of a file as follows:

```
>>> my_file =  open('file_1.txt', 'r')
>>> lines = my_file.readlines()
>>> for line in lines:
        print(line)
```

Write to a File

- We call the **method**: write
- This method takes a string, and writes it to a file, and, then, it returns the number of characters written.

```
>>> my_file_2 =  open('file_2.txt', 'w')
>>> my_file_2.write('This is my second file'+'\n')
```

- Try writing to a file using the modes: **‘a’ and ‘x’**

Write to a File-Continue

- When there is a list of lines, we call the **method**: writelines, which stores lines into a file.
- This can be called as follows:

```
>>> with open('file_5.txt','x') as file_write:
        file_write.writelines(['Python\n','Java\n','C++\n'])
```

- Try writing to a file using the modes: **‘a’ and ‘w’**

Reading from Files in a Directory

- This can be done using the **package** “os” and the **function** “listdir” as follows:

```
>>> import os

>>> os.listdir('<target directory>')
```

- To list all files with a specific extension use the **endswith method** as follows:

```
>>> [file for file in os.listdir('<directory>') if file.endswith('.txt')]
```

XML Documents

- Extensible Markup Language (XML) is a markup language
- It encodes documents by defining a set of rules that can be both machine and human readable
- Here is an example of raw XML:

```
<message>
      <from>0599000001</from>
      <to>0599000002</to>
      <body>Happy birthday my friend</body>
</message>
```

XML Documents Parsing

- XML can be parsed using **ElementTree** package.
- ElementTree supports **XPath**
- Here is parsing a simple example:

```
import xml.etree.ElementTree as ET
tree = ET.parse('message.xml')
root = tree.getroot()
messages = []
for message in root.findall('message'):
        message_temp = {}
        message_temp['from'] = message.find('from').text
        message_temp['to'] = message.find('to').text
        message_temp['body'] = message.find('body').text
        messages.append(message_temp)
```

- Let's do more practices on data found on my GitHub

JSON Data

- JSON stands for JavaScript Object Notation
- It is an open standard format
- It holds key-value pairs and array data types
- Here is an example of a raw JSON object:

```
{
    "message": {
        "from": "0599000001",
        "to": "0599000002",
        "body": "Happy birthday my friend"
    }
}
```

XML vs JSON

XML	JSON
Data is stored as a tree structure	Data is stored in key-value pairs
The size of document is larger	JSON is shorter, so, the size of file is smaller
Slow in parsing, leading to slower data transmission	Faster parsing, and hence, faster transfer of data
It does not support array directly	It supports arrays
It supports many complex data types including charts, images and other non-primitive data types.	It only supports strings, numbers, arrays boolean and objects. But objects can only contain primitive types.
It is less human readable	It is more human readable

Parsing JSON Data

- This can be done by the Python package JSON
- Here is a basic example:

```
import json

data =  '{"name":"Samer", "job": "Accountant",
"company":"Jawwal", "city":"Nablus"}'

data_parsed = json.loads(data)

print("This is {} from
{}".format(data_parsed["name"],data_parsed["city"]))
```

- Let's do more practices on data found on my GitHub

Challenge

Collect you own data stored in an XML document and do the following:

- Parse the XML document
- Convert it to json format and saves it in a json file
- Again, parse the original XML and the new JSON files, and do some comparisons between the two parsers in terms of size and time

Note: It is encouraging to use file with “big” sizes that could be handled by your machine’s memory.