File Handling

File handling is an import part of programming which allows us to create, read, update and delete files. In Python to handle data we use *open()* built-in function.

```
# Syntax
open('filename', mode) # mode(r, a, w, x, t,b)  could be to read, write,
update
```

- "r" - Read - Default value. Opens a file for reading, it returns an error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists
- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

Opening Files for Reading

The default mode of *open* is reading, so we do not have to specify 'r' or 'rt'. I have created and saved a file named reading_file_example.txt in the files directory. Let us see how it is done:

```
f = open('./files/reading_file_example.txt')
print(f) # <_io.TextIOWrapper name='./files/reading_file_example.txt'
mode='r' encoding='UTF-8'>
```

As we can see in the example above, I printed the opened file and it gave some information about it. Opened file has different reading methods: *read()*, *readline*, *readlines*. An opened file has to be closed with *close()* method.

- ***read()*:** read the whole text as string. If we want to limit the number of characters we want to read, we can limit it by passing int value to the *read(number)* method.

```
f = open('./files/reading_file_example.txt')
txt = f.read()
print(type(txt))
print(txt)
f.close()
# output
<class 'str'>
This is an example to show how to open a file and read.
This is the second line of the text.
```

Instead of printing all the text, let us print the first 10 characters of the text file.

```
f = open('./files/reading_file_example.txt')
txt = f.read(10)
print(type(txt))
print(txt)
f.close()
# output
<class 'str'>
This is an
```

- ***readline()*:** read only the first line

```
f = open('./files/reading_file_example.txt')
line = f.readline()
print(type(line))
print(line)
f.close()
# output
<class 'str'>
This is an example to show how to open a file and read.
```

- *readlines()*: read all the text line by line and returns a list of lines

```
f = open('./files/reading_file_example.txt')
lines = f.readlines()
print(type(lines))
print(lines)
f.close()
# output
<class 'list'>
['This is an example to show how to open a file and read.\n', 'This is the
second line of the text.']
```

Another way to get all the lines as a list is using *splitlines()*:

```
f = open('./files/reading_file_example.txt')
lines = f.read().splitlines()
print(type(lines))
print(lines)
f.close()
# output
<class 'list'>
['This is an example to show how to open a file and read.', 'This is the
second line of the text.']
```

After we open a file, we should close it. There is a high tendency of forgetting to close them. There is a new way of opening files using *with* - closes the files by itself. Let us rewrite the the previous example with the *with* method:

```
with open('./files/reading_file_example.txt') as f:
    lines = f.read().splitlines()
    print(type(lines))
    print(lines)
# output
<class 'list'>
['This is an example to show how to open a file and read.', 'This is the
second line of the text.']
```

Opening Files for Writing and Updating

To write to an existing file, we must add a mode as parameter to the *open()* function:

- "a" - append - will append to the end of the file, if the file does not it creates a new file.
- "w" - write - will overwrite any existing content, if the file does not exist it creates.

Let us append some text to the file we have been reading:

```
with open('./files/reading_file_example.txt','a') as f:
```

```
    f.write('This text has to be appended at the end')
```

The method below creates a new file, if the file does not exist:

```
with open('./files/writing_file_example.txt','w') as f:
    f.write('This text will be written in a newly created file')
```

Deleting Files

We have seen in previous section, how to make and remove a directory using *os* module. Again now, if we want to remove a file we use *os* module.

```
import os
os.remove('./files/example.txt')
```

If the file does not exist, the remove method will raise an error, so it is good to use a condition like this:

```
import os
if os.path.exists('./files/example.txt'):
    os.remove('./files/example.txt')
else:
    print('The file does not exist')
```

File Types

File with txt Extension

File with *txt* extension is a very common form of data and we have covered it in the previous section. Let us move to the JSON file

File with json Extension

JSON stands for JavaScript Object Notation. Actually, it is a stringified JavaScript object or Python dictionary.

Example:

```
# dictionary
person_dct= {
    "name":"Prasanta",
    "country":"India",
    "city":"Bangalore",
    "skills":["JavaScrip", "React","Python"]
}
# JSON: A string form a dictionary
person_json = "{'name': 'Prasanta', 'country': 'India', 'city':
'Bangalore', 'skills': ['JavaScrip', 'React', 'Python']}"

# we use three quotes and make it multiple line to make it more readable
person_json = '''{
    "name":"Prasanta",
    "country":"India",
    "city":"Bangalore",
```

```
    "skills":["JavaScrip", "React","Python"]
}'''
```

Changing JSON to Dictionary

To change a JSON to a dictionary, first we import the json module and then we use *loads* method.

```
import json
# JSON
person_json = '''{
    "name": "Prasanta",
    "country": "India",
    "city": "Bangalore",
    "skills": ["JavaScrip", "React", "Python"]
}'''
# let's change JSON to dictionary
person_dct = json.loads(person_json)
print(type(person_dct))
print(person_dct)
print(person_dct['name'])
# output
<class 'dict'>
{'name': 'Prasanta', 'country': 'India', 'city': 'Bangalore', 'skills':
['JavaScrip', 'React', 'Python']}
```

Changing Dictionary to JSON

To change a dictionary to a JSON we use *dumps* method from the json module.

```
import json
# python dictionary
person = {
    "name": "Prasanta",
    "country": "India",
    "city": "Bangalore",
    "skills": ["JavaScrip", "React", "Python"]
}
# let's convert it to  json
person_json = json.dumps(person, indent=4) # indent could be 2, 4, 8. It
beautifies the json
print(type(person_json))
print(person_json)
# output
# when we print it, it does not have the quote, but actually it is a string
# JSON does not have type, it is a string type.
<class 'str'>
{
    "name": "Prasanta",
    "country": "India",
    "city": "Bangalore",
    "skills": [
        "JavaScrip",
        "React",
        "Python"
    ]
}
```

Saving as JSON File

We can also save our data as a json file. Let us save it as a json file using the following steps. For writing a json file, we use the json.dump() method, it can take dictionary, output file, ensure_ascii and indent.

```
import json
# python dictionary
person = {
    "name": "Prasanta",
    "country": "India",
    "city": "Bangalore",
    "skills": ["JavaScrip", "React", "Python"]
}
with open('./files/json_example.json', 'w', encoding='utf-8') as f:
    json.dump(person, f, ensure_ascii=False, indent=4)
```

In the code above, we use encoding and indentation. Indentation makes the json file easy to read.

File with csv Extension

CSV stands for comma separated values. CSV is a simple file format used to store tabular data, such as a spreadsheet or database. CSV is a very common data format in data science.

Example:

```
"name","country","city","skills"
"Prasanta","India","Bangalore","JavaScript"
```

Example:

```
import csv
with open('./files/csv_example.csv') as f:
    csv_reader = csv.reader(f, delimiter=',') # w use, reader method to
read csv
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are :{", ".join(row)}')
            line_count += 1
        else:
            print(
                f'\t{row[0]} is a teachers. He lives in {row[1]},
{row[2]}.')
            line_count += 1
    print(f'Number of lines:  {line_count}')
# output:
Column names are :name, country, city, skills
        Prasanta is a teacher. He lives in India, Bangalore.
Number of lines:  2
```

File with xlsx Extension

To read excel files we need to install *xlrd* package. We will cover this after we cover package installing using pip.

```
import xlrd
excel_book = xlrd.open_workbook('sample.xls)
print(excel_book.nsheets)
print(excel_book.sheet_names)
```

File with xml Extension

XML is another structured data format which looks like HTML. In XML the tags are not predefined. The first line is an XML declaration. The person tag is the root of the XML. The person has a gender attribute. **Example:XML**

```
<?xml version="1.0"?>
<person gender="female">
  <name>Prasanta</name>
  <country>India</country>
  <city>Bangalore</city>
  <skills>
    <skill>JavaScrip</skill>
    <skill>React</skill>
    <skill>Python</skill>
  </skills>
</person>
```

```
import xml.etree.ElementTree as ET
tree = ET.parse('./files/xml_example.xml')
root = tree.getroot()
print('Root tag:', root.tag)
print('Attribute:', root.attrib)
for child in root:
    print('field: ', child.tag)
# output
Root tag: person
Attribute: {'gender': 'male'}
field: name
field: country
field: city
field: skills
```