09 File handling

File handling refers to the process of performing operations on a file such as creating, opening, reading, writing and closing it, through a programming interface.

Opening a File in Python

To open a file we can use `open()` function, which requires file path and mode as arguments.

```
# Open the file and read its contents
with open('sample.txt', 'r') as file:
```

Using `with` Statement

`with open()` **as** method automatically handles closing the file once the block of code is exited, even if an error occurs.

```
>>> with open("sample.txt", "r") as file:
...     content = file.read()
...     print(content)

Hello, World!
Appended text.
```

File Modes in Python

When opening a file, we must specify the mode we want to which specifies what we want to do with the file. Here's a table of the different modes available

- `r` - **Read-only** mode; Opens the file for reading. **File must exist**; otherwise, it raises an error.
- `rb` - **Read-only** in **binary** mode; Opens the file for reading binary data. File must exist; otherwise, it raises an error.
- `r+` - **Read** and **write** mode; Opens the file for both reading and writing. File must exist; otherwise, it raises an error.
- `rb+` - **Read** and **write** in **binary** mode; Opens the file for both reading and writing binary data. File must exist; otherwise, it raises an error.
- `w` - **Write** mode; Opens the file for writing. **Creates a new** file or **truncates the existing** file.

- `wb` - **Write** in **binary** mode; Opens the file for writing binary data. Creates a new file or truncates the existing file.
- `w+` - **Write** and **read** mode; Opens the file for both writing and reading. Creates a new file or truncates the existing file.
- `wb+` - **Write** and **read** in **binary** mode; Opens the file for both writing and reading binary data. Creates a new file or truncates the existing file.
- `a` - **Append** mode; Opens the file for appending data. **Creates a new** file if it doesn't exist.
- `ab` - **Append** in **binary** mode; Opens the file for appending binary data. Creates a new file if it doesn't exist.
- `a+` - **Append** and **read** mode; Opens the file for appending and reading. Creates a new file if it doesn't exist.
- ab+ - **Append** and **read** in **binary** mode; Opens the file for appending and reading binary data. Creates a new file if it doesn't exist.
- `x` - **Exclusive creation** mode; **Creates a new file**. Raises an **error if the file already exists**.
- `xb` - **Exclusive creation** in **binary** mode; Creates a new binary file. Raises an error if the file already exists.
- `x+` - **Exclusive creation** with **read** and **write** mode; Creates a new file for reading and writing. Raises an error if the file exists.
- `xb+` - **Exclusive creation** with **read** and **write** in binary mode; Creates a new binary file for reading and writing. Raises an error if the file exists.

Reading a File

Reading a file can be achieved by `file.read()` which reads the entire content of the file. After reading the file we can close the file using file.close() which closes the file after reading it, which is necessary to free up system resources.

Example: Assume sample.txt file already exist with some content.

```
>>> file = open("sample.txt", "r")
>>> content = file.read()
>>> print(content)
Hello world
WelcometoPython
123 456
>>> file.close()
```

Line-by-Line Reading in Python

We may want to read a file line by line, especially for large files where reading the entire content at once is not practical. It is done with following two methods.

- **for line in file**: Iterates over each line in the file.
- **line.strip()**: Removes any leading or trailing whitespace, including newline characters.

```
file = open("sample.txt", "r")
# Read each line one by one
for line in file:
    print(line.strip())  # .strip() to remove newline characters
file.close()

# Output
Hello World
Hello WelcometoPython
```

Using `readline()`

file.readline() reads one line at a time. while line continues until there are no more lines to read.

```
file = open("sample.txt", "r")
# Read the first line
line = file.readline()
while line:
    print(line.strip())
    line = file.readline()  # Read the next line
file.close()

# Output
Hello World
Hello WelcometoPython
```

Reading Binary Files in Python

Binary files store data in a format not meant to be read as text. These can include images, executablesor any non-text data. We are using `open("sample.txt", "rb")`

```
file = open("sample.txt", "rb")
content = file.read()
print(content)
file.close()
```

```
# Output
b'Hello world\r\nWelcometoPython\r\n123 456'
```

Reading Specific Parts of a File

Sometimes, we may only need to read a specific part of a file, such as the first few bytes, a specific line, or a range of lines.

```
file = open("sample.txt", "r")
# Read the first 10 bytes
content = file.read(10)
print(content)
file.close()

# Output
Hello World
```

Reading CSV Files

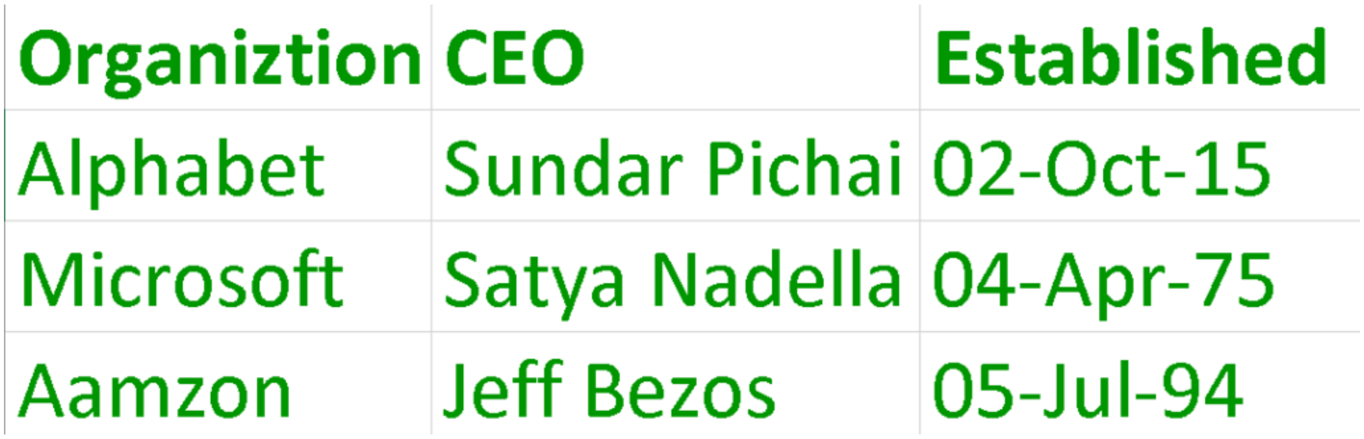
A **CSV (Comma Separated Values)** file is a form of plain text document that uses a particular format to organize tabular information. It is the most popular file format for importing and exporting spreadsheets and databases.

Reading a CSV File

There are various ways to read a CSV file in Python that use either the CSV module or the pandas library.

- **csv Module**: The CSV module is one of the modules in Python that provides classes for reading and writing tabular information in CSV file format.
- **pandas Library**: The pandas library is one of the open-source Python libraries that provide high-performance, convenient data structures and data analysis tools and techniques for Python programming.

Consider the below CSV file named **Giants.CSV**



Organiztion	CEO	Established
Alphabet	Sundar Pichai	02-Oct-15
Microsoft	Satya Nadella	04-Apr-75
Aamzon	Jeff Bezos	05-Jul-94

Reading a CSV File Format in Python

Example

```
import csv
with open('Giants.csv', mode ='r')as file:
  csvFile = csv.reader(file)
  for lines in csvFile:
        print(lines)

# Output
['Organization', 'CEO', 'Established']
['Alphabet', 'Sundar Pichai', '02-Oct-15']
['Microsoft', 'Satya Nadella', '04-Apr-75']
['Amazon', 'Jeff Bezos', '05-Jul-94']
```

It is similar to the previous method, the CSV file is first opened using the `open()` method then it is read by using the `DictReader` class of csv module which works like a regular reader but maps the information in the CSV file into a dictionary.

Example

```
import csv
with open('Giants.csv', mode ='r') as file:
       csvFile = csv.DictReader(file)
       for lines in csvFile:
            print(lines)
```

Using pandas.read_csv() method

The below code reads the CSV file and stores it as a DataFrame using the `pandas.read_csv()` function. Finally, it prints the entire DataFrame, which provides a structured and tabular representation of the CSV data.

Example

```
import pandas
csvFile = pandas.read_csv('Giants.csv')
print(csvFile)

# Output
Organization            CEO Established
0   Alphabet  Sundar Pichai   02-Oct-15
1   Microsoft  Satya Nadella   04-Apr-75
2   Amazon     Jeff Bezos   05-Jul-94
```

Read JSON files

The full form of JSON is JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called JSON.

Example

```
import json

# Open and read the JSON file
with open('data.json', 'r') as file:
    data = json.load(file)

# Print the data
print(data)
```

Parse JSON – How to Read a JSON File

Python has a built-in package called JSON, which can be used to work with JSON data. It's done by using the JSON module, which provides us with a lot of methods which among `loads()` and `load()` methods are gonna help us to read the JSON file.

Loading a JSON File in Python

Here we are going to read a JSON file named **data.json**

```
/*data.json*/
{
  "emp_details": [
    {
      "emp_name": "Shubham",
      "email": "ksingh.shubh@gmail.com",
      "job_profile": "intern"
    },
    {
      "emp_name": "Gaurav",
      "email": "gaurav.singh@gmail.com",
      "job_profile": "developer"
    },
    {
      "emp_name": "Nikhil",
      "email": "nikhil@example.org",
      "job_profile": "Full Time"
    }
  ]
}
```

Deserialize a JSON String to an Object

The Deserialization of JSON means the conversion of JSON objects into their respective Python objects. The load()/loads() method is used for it.

```
import json

f = open('data.json')
data = json.load(f)

for i in data['emp_details']:
    print(i)

f.close()

# Output
{'emp_name': 'Shubham', 'email': 'ksingh.shubh@gmail.com', 'job_profile':
'intern'},
{'emp_name': 'Gaurav', 'email': 'gaurav.singh@gmail.com', 'job_profile':
'developer'},
{'emp_name': 'Nikhil', 'email': 'nikhil@example.org', 'job_profile': 'Full
Time'}
```

Python Read JSON String

This example shows reading from a JSON string using json.loads() method.

```
import json

j_string = '{"name": "Bob", "languages": "English"}'

y = json.loads(j_string)
print("JSON string = ", y)
```

Writing to a File

Writing to a file in Python means saving data generated by your program into a file on your system.

```
>>> file = open("sample.txt", "w")
>>> file.write("Hello, World!")
>>> file.close()
```

Creating a File

Example

```
# Write mode: Creates a new file or truncates an existing file
with open("sample.txt", "w") as f:
    f.write("Created using write mode.")

f = open("sample.txt","r")
print(f.read())

# Append mode: Creates a new file or appends to an existing file
with open("sample.txt", "a") as f:
    f.write("Content appended to the file.")

f = open("sample.txt","r")
print(f.read())

# Exclusive creation mode: Creates a new file, raises error if file exists
try:
    with open("sample.txt", "x") as f:
        f.write("Created using exclusive mode.")
except FileExistsError:
    print("Already exists.")

# Output
Created using write mode.
Created using write mode. Content appended to the file.
Already exists.
```

Writing lines to a File

writelines(): Allows us to write a list of string to the file in a single call.

```
# Writing multiple lines to an existing file using writelines()
s = ["First line of text.\n", "Second line of text.\n", "Third line of
text.\n"]

with open("sample.txt", "w") as f:
    f.writelines(s)

f = open("file1.txt","r")
print(f.read())

# Output
Written to the file.
First line of text.
Second line of text.
Third line of text.
```

Writing to a Binary File

When dealing with non-text data (e.g., images, audio, or other binary data), Python allows you to write to a file in binary mode.

```
# Writing binary data to a file
bin = b'\x00\x01\x02\x03\x04'

with open("file.bin", "wb") as f:
    f.write(bin)

f = open("file.bin","r")
print(f.read())
```

Write to CSV Files

The `csv` module provides the `csv.writer()` function to write to a CSV file.

Let's look at an example.

```python
import csv
with open('protagonist.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["SN", "Movie", "Protagonist"])
    writer.writerow([1, "Lord of the Rings", "Frodo Baggins"])
    writer.writerow([2, "Harry Potter", "Harry Potter"])
```

When we run the above program, a **protagonist.csv** file is created with the following content

```
SN,Movie,Protagonist
1,Lord of the Rings,Frodo Baggins
2,Harry Potter,Harry Potter
```

Writing JSON to a file

To write JSON to a file in Python, we can use `json.dump()` method.

Example

```python
import json

person_dict = {"name": "Bob",
"languages": ["English", "French"],
"married": True,
"age": 32
}

with open('person.txt', 'w') as json_file:
  json.dump(person_dict, json_file)
```

When you run the program, the `person.txt` file will be created. The file has following text inside it.

```json
{"name": "Bob", "languages": ["English", "French"], "married": true, "age":
32}
```

Closing a File

Closing a file is essential to ensure that all resources used by the file are properly released. `file.close()` method closes the file and ensures that any changes made to the file are saved.

```
>>> file = open("sample.txt", "r")
>>> # Perform file operations
>>> file.close()
```

Handling Exceptions When Closing a File

It’s important to handle exceptions to ensure that files are closed properly, even if an error occurs during file operations.

```
>>> try:
...     file = open("sample.txt", "r")
...     content = file.read()
...     print(content)
... finally:
...     file.close()
...
Hello, World!
Appended text.
```

Advantages of File Handling

- **Versatility**: File handling in Python allows us to perform a wide range of operations, such as creating, reading, writing, appending, renaming and deleting files.
- **Flexibility**: File handling in Python is highly flexible, as it allows us to work with different file types (e.g. text files, binary files, CSV files , etc.) and to perform different operations on files (e.g. read, write, append, etc.).
- **User – friendly**: Python provides a user-friendly interface for file handling, making it easy to create, read and manipulate files.
- **Cross-platform**: Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

Disadvantages of File Handling

- **Error-prone**: File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).
- **Security risks**: File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.
- **Complexity**: File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure

that files are handled properly and securely.
- **Performance**: File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

Directory and Files Management

A directory is a collection of **files** and **subdirectories**. A directory inside a directory is known as a subdirectory.

Python has the `os` module that provides us with many useful methods to work with directories (and files as well).

Get Current Directory

We can get the present working directory using the `getcwd()` method of the `os` module.

For example

```
import os

print(os.getcwd())

# Output: C:\Program Files\PyScripter
```

Here, `getcwd()` returns the current directory in the form of a string.

Changing Directory

In Python, we can change the current working directory by using the `chdir()` method.

The path that we want to change into must be supplied as a string to this method. And we can use both the forward-slash `/` or the backward-slash `\` to separate the path elements.

Let's see an example

```
import os

# change directory
os.chdir('C:\\Python33')

print(os.getcwd())

Output: C:\Python33
```

List Directories and Files

All files and sub-directories inside a directory can be retrieved using the `listdir()` method.

This method takes in a path and returns a list of subdirectories and files in that path.

If no path is specified, it returns the list of subdirectories and files from the current working directory.

Example

```
import os

print(os.getcwd())
# C:\Python33

# list all sub-directories
os.listdir()
['DLLs',
'Doc',
'include',
'Lib',
'libs',
'LICENSE.txt',
'NEWS.txt',
'python.exe',
'pythonw.exe',
'README.txt',
'Scripts',
'tcl',
'Tools']

os.listdir('G:\\')
['$RECYCLE.BIN',
'Movies',
'Music',
'Photos',
'Series',
'System Volume Information']
```

Making a New Directory

In Python, we can make a new directory using the `mkdir()` method.

This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

```
os.mkdir('test')

os.listdir()
['test']
```

Renaming a Directory or a File

The `rename()` method can rename a directory or a file.

For renaming any directory or file, `rename()` takes in two basic arguments:

- the old name as the first argument
- the new name as the second argument.

Example

```
import os

os.listdir()
['test']

# rename a directory
os.rename('test','new_one')

os.listdir()
['new_one']
```

Here, `'test'` directory is renamed to `'new_one'`

Removing Directory or File

In Python, we can use the `remove()` method or the `rmdir()` method to remove a file or directory.

```
import os

# delete "sample.txt" file
os.remove("sample.txt")
```

```
# delete the empty directory "mydir"
os.rmdir("mydir")
```

In order to remove a non-empty directory, we can use the `rmtree()` method inside the `shutil` module.

```
import shutil

# delete "mydir" directory and all of its contents
shutil.rmtree("mydir")
```