Chapter-3
File Handling

File Handling

- Programming often involves reading information from files and writing information into files.
- Reading and writing files can lead to exceptions when the file specified cannot be found.
- There are many other situations in which exceptions can occur
- text vs binary
  - Store human-readable text (strings).
  - Store raw bytes (binary format).

Opening a file

- Reading information from and writing information to files is a common task in programming.
- File handling in Python allows you to **create, read, write, and manipulate files.**
- It's essential when working with data storage, logging, or processing external files like .txt, .csv, .json, etc.
- Python supports the opening of a file using the **open()** function.
- It returns a file object that provides methods for reading, writing, and closing files.
- **Returns**
  - A **file object** if the file is opened successfully.
  - Raises an **exception** if the file doesn’t exist (in read mode).

Opening a file

- The **open()** function takes two parameters; **Filename, and Mode**.
- There are four different methods (modes) for opening a file:
  - "x" - Create - Creates the specified file, returns an error if the file exists
  - "w" - Write - Opens a file for writing, creates the file if it does not exist
  - "r" - Read - Default value. Opens a file for reading, error if the file does not exist
  - "a" - Append - Opens a fille for appending, creates the file if it does not exist
- In addition you can specify if the file should be handled as binary or text mode
  - "t" - Text - Default value. Text mode (It reads/writes strings)
  - "b" - Binary - Binary mode (e.g. images) (It reads/writes bytes)

Read Data from a File

- Python provides functions that can be called on a file object for reading the contents of a file:
- The **read()** function reads the contents of a file and returns a string.
- The **readline()** function reads the next line in a file and returns a string.
- The **readlines()** function reads the individual lines of a file and returns a string list containing all the lines of the file in order.

```
f = open("myfile.txt", "r")

data = f.read()

print(data)

f.close()
```

Reads the entire file:

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

With character limit:

```
f = open("sample.txt", "r")
print(f.read(10))   # Reads first 10 characters
f.close()
```

Reads one line at a time:

```
f = open("sample.txt", "r")
print(f.readline())  # Reads first line
print(f.readline())  # Reads second line
f.close()
```

Reads all lines and returns a list:

```
f = open("sample.txt", "r")
lines = f.readlines()
print(lines)
f.close()
```

Writing to files

- To create a New file in Python, use the **open()** method, with one of the following parameters:
  - "x" - Create - Creates the specified file, returns an error if the file exists
  - "w" - Write - Opens a file for writing, creates the file if it does not exist
  - "a" - Append - Opens a file for appending, creates the file if it does not exist

```
f = open("myfile.txt", "x")

data = "Helo World"

f.write(data)

f.close()
```

Writes a string to the file:

```
f = open("output.txt", "w")
f.write("Hello, Python!\n")
f.write("File handling is easy.")
f.close()
```

Note: If the file exists, it **overwrites** it.

Writes multiple lines from a list:

```
lines = ["First line\n", "Second line\n", "Third line\n"]
f = open("output.txt", "w")
f.writelines(lines)
f.close()
```

Appends new data without overwriting:

```
f = open("output.txt", "a")
f.write("\nThis is an extra line.")
f.close()
```

Logging Data to File

- Typically, you want to write multiple data to the file
- e.g., assume you read some temperature data at regular intervals and then you want to save the temperature values to a File.

```
data = [1.6, 3.4, 5.5, 9.4]

f = open("myfile.txt", "x")

for value in data:
    record = str(value)
    f.write(record)
    f.write("\n")

f.close()
```

```
f = open("myfile.txt", "r")

for record in f:
    record = record.replace("\n", "")
    print(record)

f.close()
```

Using with Statement

- Using **with open()** is recommended because:
  - Automatically closes the file
  - Prevents resource leaks.

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

```
with open("output.txt", "w") as f:
    f.write("Using with-statement is safe!")
```

Handling File Exceptions

- When dealing with files, errors like **file not found** are common.

```
try:
    f = open("unknown.txt", "r")
    print(f.read())
except FileNotFoundError:
    print("The file does not exist.")
finally:
    try:
        f.close()
    except:
        pass
```

```
# Create and write to file
with open("demo.txt", "w") as f:
    f.write("Python is amazing!\n")
    f.write("File handling is simple.\n")

# Read file content
with open("demo.txt", "r") as f:
    print("File Content:")
    print(f.read())

# Append more content
with open("demo.txt", "a") as f:
    f.write("Appending new data...\n")

# Read again
with open("demo.txt", "r") as f:
    print("\nUpdated File Content:")
    print(f.read())
```

Some sample problems

- You want to redirect the output of the print() function to a file.
- Solution: Use the file keyword argument to print(), like this:

```
with open('somefile.txt', 'rt') as f:
    print('Hello World!', file=f)
```

- You want to write data to a file, but only if it doesn't already exist on the filesystem

```
>>> with open('somefile', 'wt') as f:
...     f.write('Hello\n')
...
>>> with open('somefile', 'xt') as f:
...     f.write('Hello\n')
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileExistsError: [Errno 17] File exists: 'somefile'
```

CSV File Handling

- CSV (Comma-Separated Values) files store tabular data in plain text.
- Each line : one row
- Each value : separated by a comma (, by default, but it can be ;, \t, etc.)
- Commonly used for data exchange between spreadsheets, databases, and Python.

```
Name,Age,Marks
John,20,85
Alice,22,90
Bob,19,78
```

CSV File Handling

- CSV files are opened just like text files

```
f = open("students.csv", "r")   # Read mode
f = open("students.csv", "w")   # Write mode
f = open("students.csv", "a")   # Append mode
f.close()
```

Reading CSV Files Using csv Module

- Python's **csv module** provides two main classes:
- **csv.reader:** Reads data **row by row**
- **csv.DictReader:** Reads data into **dictionaries**

```
import csv

with open("students.csv", "r") as f:
    csv_reader = csv.reader(f)

    for row in csv_reader:
        print(row)
```

```
import csv

with open("students.csv", "r") as f:
    csv_reader = csv.reader(f)
    next(csv_reader)  # Skip the first row (header)

    for row in csv_reader:
        print(row)
```

Reading CSV Files Using csv Module

- Instead of returning lists, this converts **each row into a dictionary** using the **header names as keys**

```
import csv

with open("students.csv", "r") as f:
    csv_reader = csv.DictReader(f)
    for row in csv_reader:
        print(row)
```

```
['Name', 'Age', 'Marks']
['John', '20', '85']
['Alice', '22', '90']
['Bob', '19', '78']
```

```
{'Name': 'John', 'Age': '20', 'Marks': '85'}
{'Name': 'Alice', 'Age': '22', 'Marks': '90'}
{'Name': 'Bob', 'Age': '19', 'Marks': '78'}
```

Reading CSV Files Using csv Module

- Python provides:
  - **csv.writer()** → Writes rows as **lists**.
  - **csv.DictWriter()** → Writes rows as **dictionaries**.

```
import csv
data = [
   ["Name", "Age", "Marks"],
   ["John", 20, 85],
   ["Alice", 22, 90],
   ["Bob", 19, 78]
]
with open("output.csv", "w") as f:
   writer = csv.writer(f)
   writer.writerows(data)  # Writes all rows at once
print("CSV file written successfully!")
```

Reading CSV Files Using csv Module

```
import csv

data = [
    {"Name": "John", "Age": 20, "Marks": 85},
    {"Name": "Alice", "Age": 22, "Marks": 90},
    {"Name": "Bob", "Age": 19, "Marks": 78}
]

with open("output.csv", "w", newline="") as f:
    fieldnames = ["Name", "Age", "Marks"]
    writer = csv.DictWriter(f, fieldnames=fieldnames)

    writer.writeheader()  # Write column names
    writer.writerows(data)

print("CSV file written successfully!")
```

```
import csv

with open("students.csv", "a", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["David", 21, 88])
```

File Handling: JSON

- JSON stands for JavaScript Object Notation.
- It is a lightweight, text-based format used to store and exchange data.
- Where JSON is used?
  - Web development: Data exchange between frontend & backend.
  - APIs: Most APIs return data in JSON.
  - Databases: NoSQL databases like MongoDB store data in JSON-like format.
  - Configurations: App settings, ML model parameters.

```
{
 "name": "Alice",
 "age": 25,
 "is_student": true,
 "marks": [85, 90, 92],
 "address": {
  "city": "London",
  "zipcode": "E1 6AN"
 }
}
```

File Handling: JSON

```
{
  "name": "Alice",              // string
  "age": 25,                    // number (int)
  "height": 5.6,                // number (float)
  "is_student": true,           // boolean
  "hobbies": ["reading", "music", "sports"],   // array
  "scores": [85, 90, 92],       // array of numbers
  "address": {                  // nested object
    "city": "London",
    "zipcode": "E1 6AN"
  },
  "nickname": null              // null
}
```

```
{
  "name": "Alice",              # str
  "age": 25,                    # int
  "height": 5.6,                # float
  "is_student": True,           # bool
  "hobbies": ["reading", "music", "sports"],  # list of str
  "scores": [85, 90, 92],       # list of int
  "address": { "city": "London", "zipcode": "E1 6AN" },  # dict
  "nickname": None              # NoneType
}
```

File Handling: JSON

- **Reading JSON files**
  - You use json.load() to **read from a JSON file**.

```python
import json

# Open JSON file and load it
with open("data.json", "r") as f:
    data = json.load(f)

print(data)
print(data["name"])
print(data["marks"])
```

```json
{
  "name": "Alice",
  "age": 25,
  "marks": [85, 90, 92],
  "city": "London",
  "is_student": true
}
```

- **Writing JSON files**
  - You use json.dump() to write **Python objects to a JSON file**.

```python
import json

data = {
    "name": "Bob",
    "age": 30,
    "marks": [78, 82, 91],
    "city": "Paris",
    "is_student": False
}

# Write data to JSON file
with open("output.json", "w") as f:
    json.dump(data, f, indent=4)   # indent=4 → makes it pretty-printed
```

- **Reading JSON from a string**
- Sometimes you get JSON data from an API as a string.
- Use json.loads():

```
json_string = '{"name": "Charlie", "age": 28, "city": "Berlin"}'

data = json.loads(json_string)
print(data["city"])   # Output: Berlin
```

- **Writing JSON to a string**
  - If you want a JSON string instead of saving to a file
  - use json.dumps():

```
person = {"name": "David", "age": 35, "city": "Rome"}
json_string = json.dumps(person, indent=2)
print(json_string)
```

```python
# Analyze a log file for errors
error_count = 0
with open("log.txt", "r") as file:
    for line in file:
        if "ERROR" in line:
            error_count += 1

print(f"Total errors found: {error_count}")
```

```python
keywords = ["ERROR", "WARNING", "INFO"]
counts = {key: 0 for key in keywords}

with open("system.log", "r") as f:
    for line in f:
        for key in keywords:
            if key in line:
                counts[key] += 1

for key, count in counts.items():
    print(f"{key}: {count}")
```

```
import glob

files = glob.glob("logs/*.txt")
seen = set()
duplicates_removed = 0

for file in files:
    with open(file, "r") as f:
        lines = f.readlines()

    with open(file, "w") as f:
        for line in lines:
            if line not in seen:
                f.write(line)
                seen.add(line)
            else:
                duplicates_removed += 1

print(f"Removed {duplicates_removed} duplicate lines.")
```

```python
import csv

# Suppose text file has names, CSV has marks. Merge them.
names = {}
with open("names.txt", "r") as f:
    for line in f:
        student_id, name = line.strip().split(",")
        names[student_id] = name

merged_data = []
with open("marks.csv", "r") as f:
    reader = csv.reader(f)
    header = next(reader)
    header.insert(1, "Name")  # add name column
    for row in reader:
        student_id = row[0]
        name = names.get(student_id, "Unknown")
        row.insert(1, name)
        merged_data.append(row)

with open("final_output.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(merged_data)

print("Merged text and CSV successfully.")
```