3-Hour File Handling Class in Python

This session dives deep into File Handling in Python with theory, demos, student activities, and advanced extensions—designed to fill a full 3-hour slot.

Long Detailed Introduction: File handling is a fundamental aspect of programming that enables applications to persist, retrieve, and manipulate data beyond their runtime. In real-world scenarios, programs often need to:

- **Save user-generated content** (e.g., documents, logs, configurations) to disk.
- **Read existing data** for processing, such as reading CSV reports, parsing configuration files, or loading cached results.
- **Update or append data** for ongoing records like user activity logs, audit trails, and incremental backups.

Under the hood, file I/O involves interaction with the operating system's filesystem. Python abstracts these operations through *file objects*, which represent an ongoing stream of bytes or text. Key considerations include:

- **Text vs. Binary modes:** Text mode (`'r'`, `'w'`, `'a'`) handles character encoding and newline conversions, while binary mode (`'rb'`, `'wb'`, `'ab'`) works with raw bytes.
- **Buffering and performance:** Python uses buffered I/O to minimize system calls; understanding `flush()` and `fsync()` helps manage when data truly hits the disk.
- **Error handling:** Files may not exist, permissions may be restricted, or disk space may run out. Robust applications handle these exceptions gracefully.
- **Resource management:** Forgetting to close file handles can lead to resource leaks. The `with` statement ensures deterministic closing even when errors occur.

By mastering file handling, you gain the ability to bridge in-memory computations with persistent storage—an essential skill for building versatile, reliable software systems.

Total Time: 180 mins

1. Introduction & File Objects (20 mins)

Content to Speak: "File handling allows programs to interact with external data stores. Files are streams of bytes or text that we can read, write, or append to. Python represents files as *file objects*."

Subtopics: 1.1 File I/O Importance (5 mins)

- **Speak:** "Why do applications need file I/O? Logging, configuration, data persistence."

1.2 Open Modes & Parameters (5 mins)

- Modes: `'r'`, `'w'`, `'a'`, `'r+'`, `encoding`, `newline`
- **Code:**

```
f = open('sample.txt', 'r', encoding='utf-8')
f2 = open('output.txt', 'w', newline='')
```

- **Speak:** "Mode determines if we read, write, or append. Encoding ensures correct character handling."

1.3 Manual Open/Close vs `with` (5 mins)

- **Code:**

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

- **Speak:** "Always close files to free resources."
- **Code with context manager:**

```
with open('sample.txt') as f:
    print(f.read())
# auto-closed
```

1.4 Quick Activity (5 mins)

- Students open a file in each mode, print `f.mode` and `f.closed`, then close and check status.

---

2. Reading Files (40 mins)

Content to Speak: "Python provides multiple methods for reading files—choose based on file size and processing needs."

2.1 `read()` Method (7 mins)

- **Code:**

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

- **Speak:** "Reads entire file into one string—easy but can exhaust memory on large files."
- **Exercise (2 mins):** Print `len(full_text)`.

2.2 `read(n)` - Partial Read (7 mins)

- **Code:**

```
with open('sample.txt') as f:
    part = f.read(20)
    print(part)
```

- **Speak:** "Reads first n characters—useful for chunked processing."
- **Exercise (2 mins):** Loop to read in 50-char chunks and count chunks.

2.3 `readline()` Method (7 mins)

- **Code:**

```
with open('sample.txt') as f:
    line = f.readline()
    while line:
        print(line.strip())
        line = f.readline()
```

- **Speak:** "Reads one line at a time. Good for line-based parsing."
- **Exercise (2 mins):** Count and print lines starting with a vowel.

2.4 `readlines()` Method (7 mins)

- **Code:**

```
with open('sample.txt') as f:
    lines = f.readlines()
    for i, ln in enumerate(lines,1): print(i, ln.strip())
```

- **Speak:** "Returns list of lines; easier iteration at the cost of memory."
- **Exercise (2 mins):** Reverse list of lines and display.

2.5 File Pointer & Positioning (7 mins)

- **Code:**

```
with open('sample.txt') as f:
    print('Pos:', f.tell())
    f.seek(10)
    print('After seek:', f.read(10))
```

- **Speak:** "`tell()` gives current position; `seek()` moves pointer. Useful for random access."
- **Exercise (2 mins):** Seek to middle (half of file size) and read a line.

2.6 Iterating Over File Object (7 mins)

- **Code:**

```
count=0
with open('sample.txt') as f:
    for line in f:
        count+=1
print('Total lines:', count)
```

- **Speak:** "`for line in f` is memory-efficient for large files."
- **Exercise (2 mins):** Find and print the longest line length.

---

3. Writing to Files (30 mins)

Content to Speak: "Writing to files creates or overwrites content. We use mode `'w'` to write and trust Python to handle buffering."

3.1 Write Mode & Truncation (5 mins)

- **Code:**

```
with open('output.txt','w') as f:
    f.write('Hello World\n')
```

- **Speak:** "Mode 'w' truncates existing file or creates new one."

3.2 Writing Multiple Lines (10 mins)

- **Code:**

```
lines=['Line1','Line2','Line3']
with open('output.txt','w') as f:
    for ln in lines: f.write(ln+'\n')
```

- **Speak:** "Use a loop to write multiple lines."

3.3 Activity: Poem Editor (10 mins)

- Students write a short poem to `poem.txt`, then reopen in write mode and replace a line.

3.4 Discussion: Atomic Writes, `flush()`, `os.fsync()` (5 mins)

- **Speak:** "Buffering may delay actual disk writes; for critical data use flush and fsync."

---

4. Appending to Files (20 mins)

Content to Speak: "Append mode `'a'` adds data to end of file without deleting existing content—ideal for logs."

4.1 Demo Append (5 mins)

- **Code:**

```
from datetime import datetime
with open('log.txt','a') as f:
    f.write(f"Log entry at {datetime.now()}\n")
```

- **Speak:** "Appends timestamped entries."

4.2 Exercise: Mini-counter (5 mins)

- Write a counter that reads last number from file and appends number+1.

4.3 Discussion: Log Rotation Strategies (5 mins)

- **Speak:** "How to archive or rotate log files to avoid huge size."

4.4 Activity: Students implement simple log-rotate by renaming and creating new file (5 mins).

---

5. File Methods & Context Manager (20 mins)

Content to Speak: "Python’s file object provides versatile methods. Use context managers to handle exceptions and closing automatically."

5.1 Overview of Methods (5 mins)

- `read()`, `readline()`, `readlines()`, `write()`, `writelines()`
- **Speak:** Brief difference and typical use-case.

5.2 Demo `writelines()` (5 mins)

- **Code:**

```
lines=['A','B','C']
with open('lines.txt','w') as f:
    f.writelines([ln+'\n' for ln in lines])
```

5.3 Activity: Convert open/close to `with` (5 mins)

- Provide old code; students refactor to use `with`.

5.4 Discussion: Exception Safety (5 mins)

- **Speak:** "Context managers ensure file.close even on errors."

---

6. Working with CSV Files (30 mins)

Content to Speak: "CSV files are comma-separated text—widely used for data exchange. Python's `csv` module makes it easy."

6.1 Write CSV with `csv.writer` (5 mins)

- **Code:**

```
import csv
data=[['Name','Age'],['Aman',25],['Seema',23]]
with open('data.csv','w',newline='') as f:
    csv.writer(f).writerows(data)
```

6.2 Read CSV with `csv.reader` (5 mins)

- **Code:**

```
with open('data.csv','r') as f:
    for row in csv.reader(f): print(row)
```

6.3 Activity: Filter Rows (10 mins)

- Students load CSV and print only rows where age > 20.

6.4 Advanced: `DictReader` / `DictWriter` (5 mins)

- **Code:**

```
with open('data.csv') as f:
    for d in csv.DictReader(f): print(d['Name'], d['Age'])
```

6.5 Discussion: Handling large CSVs, use of `pandas` later.

---

7. Advanced Topics & Extensions (20 mins)

Content to Speak: "Beyond text files, Python supports binary I/O, JSON, pickle, and file path utilities."

7.1 Binary File I/O (5 mins)

- **Code:**

```
with open('image.png','rb') as f:
    data=f.read()
print(len(data), 'bytes')
```

```
with open('copy.png','wb') as f:
    f.write(data)
```

7.2 JSON & Pickle (5 mins)

- **Code:**

```
import json, pickle
data={'a':1,'b':2}
with open('data.json','w') as f: json.dump(data,f)
with open('data.pickle','wb') as f: pickle.dump(data,f)
```

7.3 File Paths with `pathlib` (5 mins)

- **Code:**

```
from pathlib import Path
p=Path('sample.txt')
print(p.exists(), p.suffix, p.read_text())
```

7.4 Compressing Files (5 mins)

- **Code:**

```
import gzip
with open('output.txt','rb') as f_in, gzip.open('out.gz','wb') as f_out:
    f_out.writelines(f_in)
```

---

8. Practice Exercises & Review (20 mins)

Content to Speak: "Apply what you've learned with hands-on exercises and review key concepts."

8.1 Line Counter Script (5 mins)

- **Code:**

```
def count_lines(file): return sum(1 for _ in open(file))
print(count_lines('sample.txt'))
```

8.2 File Copier (5 mins)

- **Code:**

```
with open('input.txt') as src, open('copy.txt','w') as dst:
    dst.write(src.read())
```

8.3 Word Frequency (5 mins)

- **Code:**

```
from collections import Counter
words=open('sample.txt').read().split()
print(Counter(words))
```

8.4 Logger with Timestamp (5 mins)

- **Code:**

```
from datetime import datetime
msg=input('Enter log: ')
with open('log.txt','a') as f:
    f.write(f"{datetime.now()}: {msg}\n")
```

Group Debug Activity: Provide a buggy file-read script; students identify and fix errors.\ **Q&A:** Address `FileNotFoundError`, permission issues, encoding errors.

End of File Handling Session