Python Basics: Complete File Handling Tutorial

Instructor: Zhentong Ye (1235357)
Duration: 35-45 minutes
Platform: Jupyter Notebook

Course Outline - Progressive Difficulty Structure

Level 1: Foundation (12 minutes)

1. **What is File Handling?** - Understanding the basics
2. **Your First File** - Simple open and close
3. **The Magic Word: 'with'** - Safe file handling
4. **Basic Reading** - read() method

Level 2: Essential Skills (10 minutes)

5. **Line by Line Reading** - readline() and iteration
6. **Reading Multiple Lines** - readlines() method
7. **Your First Write** - write() method
8. **Adding to Files** - append mode

Level 3: Real-World Skills (8 minutes)

9. **When Things Go Wrong** - Exception handling
10. **File Modes Explained** - r, w, a, x modes
11. **Text Encoding** - UTF-8 and character sets

Level 4: Professional Applications (10 minutes)

12. **Working with CSV Files** - Structured data
13. **Text File Processing** - Real applications
14. **Best Practices** - Production-ready code

Level 5: Hands-On Practice (5 minutes)

15. **Progressive Challenges** - From simple to advanced

Learning Objectives

By the end of this tutorial, you will:

- Master basic file operations (open, read, write, close)
- Handle errors gracefully in file operations
- Work with different file formats (text, CSV, JSON)
- Apply file handling to real-world problems
- Follow Python best practices for file handling

Level 5: Hands-On Practice - Progressive Challenges

This level is intentionally student-driven. Work through each part in order, documenting your reasoning and referencing the files you generated earlier in the notebook.

- **Part 1: Multiple Choice Checkpoint** – interpret real artifacts to choose the most defensible answer.
- **Part 2: Fill in the Blanks** – support each blank with evidence you can point to.
- **Part 3: Hands-On Practice** – complete the coding challenges from beginner to expert.

> Tip: Resist the temptation to query AI tools. Instead, inspect the files you created, run targeted snippets, and justify every choice in your own words.

Part 1: Multiple Choice Checkpoint

Select the most defensible answer for each scenario. The options intentionally look similar—inspect the actual files produced in earlier levels before you decide.

1. **Reviewing `practice_files/sample.txt` without leaving the file open**
   - A. Open the file with `open(..., 'r')` and trust garbage collection to close it eventually.
   - B. Wrap the read in `with open('practice_files/sample.txt', 'r', encoding='utf-8') as reader:`.
   - C. Read the file via `os.path` utilities because they auto-close handles.
   - D. Use `open(..., 'r+')` so you can read and close in one call.
2. **Confirming the first line in `practice_files/sample.txt` after running the Level 1 demo**
   - A. `Python lets you handle files.`
   - B. `Welcome to Python File Handling!`
   - C. `This is line 2 of our sample file.`
   - D. `Line 3 contains some numbers: 123, 456, 789.`
3. **Counting log levels in `data/application.log` during the Level 2 exercise**
   - A. Convert the entire file to lowercase once and call `.count('error')`, `.count('warning')`, and `.count('info')`.
   - B. Iterate over each line inside a `with` block, check `'INFO'`, `'WARNING'`, and `'ERROR'` separately, and increment dedicated counters.
   - C. Load the log with `csv.reader`, treat each line as a row, and read the second column as the log level.
   - D. Use `json.load` so you can access level names directly.
4. **Understanding the effect of `.strip()` in the sample file cleanup**
   - A. It removes the trailing newline so `Clean first line:` prints without an extra blank line.
   - B. It alphabetically sorts the characters on each line.
   - C. It slices away the first three characters of every string.
   - D. It converts the text to uppercase for display.
5. **Inspecting `output/no_newlines.txt` after writing the grocery list**
   - A. The file contains the single string `EggsFlourSugar`.
   - B. The file shows three lines separated by `\n`.
   - C. The file stores the list literal `['Eggs', 'Flour', 'Sugar']`.
   - D. The file is empty because `writelines` requires a newline argument.
6. **Verifying the final line in `output/diary.txt` once append mode finishes**
   - A. `Day 1: Started learning file handling`
   - B. `Day 2: Learned about append mode`
   - C. `Day 3: Getting more confident!`
   - D. `Day 4: Practiced writing CSV files`
7. **Interpreting the return value of `safe_read_file('practice_files/missing.txt')`**
   - A. It returns the string `'Missing file'`.
   - B. It returns `None` after printing an explanatory message.
   - C. It raises a `FileNotFoundError` back to the caller.
   - D. It returns an empty dictionary.
8. **Tracking configuration changes before saving `output/updated_config.json`**
   - A. `config['application']['debug_mode']` stays `False`.
   - B. A new flag `config['features']['new_feature'] = True` is added to the configuration.
   - C. `config['database']['host']` switches to `'remote-server'`.
   - D. `config['logging']['level']` is downgraded to `'DEBUG'`.
9. **Summing all sales values in `data/sales_data.csv`**
   - A. $2,837.47$ — B.3,037.46
   - C. $3,250.00$ — D.3,333.33
10. **Identifying the highest-grossing category from the CSV analysis**

- A. Kitchen
- B. Electronics
- C. Education
- D. Furniture

11. **Reading the `time_range` returned by `analyze_log_file('data/application.log')`**

- A. `('2024-01-15 09:00:00', '2024-01-15 11:00:04')`
- B. `('2024-01-15 09:15:23', '2024-01-15 10:45:41')`
- C. `('2024-01-15 09:30:45', '2024-01-15 10:30:18')`
- D. `('2024-01-15 10:00:33', '2024-01-15 10:15:56')`

12. **Explaining why `encoding='utf-8'` is specified in the Unicode example**

- A. It forces Python to drop any emoji characters so the file stays ASCII.
- B. It ensures emojis and non-Latin characters such as `你好` survive the write/read cycle.
- C. It automatically compresses the file to save disk space.
- D. It converts all numbers to floats before saving.

Part 2: Fill in the Blanks

Complete each statement by providing the missing phrase. Cite the evidence you used (file name, line number, or snippet) in your notes.

1. The setup cell calls `os.makedirs(directory, _____ )` so rerunning it never raises an error when a folder already exists.
2. The generated `sales_data.csv` header lists the second column as `_____`.
3. Line 3 of `practice_files/sample.txt` records the numbers `_____`.
4. After applying `.strip()` to `first_line`, the trailing `_____` character disappears.
5. The log warning about disk capacity reports only `_____` of free space.
6. The counters `info_count`, `warning_count`, and `error_count` each start at `_____` before the loop.
7. When `safe_read_file` cannot locate a file, it prints a message and returns `_____`.
8. The base diary entry begins with the title `_____`.
9. Once append mode finishes, the new final line in `output/diary.txt` reads `_____`.
10. Inside the CSV dictionary loop, totals accumulate with `float(sale['_____'])`.
11. The database configuration retains a `timeout` value of `_____` seconds.
12. `output/unicode_test.txt` includes the greeting `Chinese: _____`.

Part 3: Hands-On Practice (Coding Challenges)

Challenge 1: Personal Notes System (Beginner)

Create a simple note-taking system:

```
# Your task: Complete this function
from datetime import datetime

def create_note(filename, title, content):
    """Create a note file with title and content"""
    # Step 1: Get current timestamp
    # Step 2: Format the note with title, timestamp, and content
    # Step 3: Write to file
    # Step 4: Confirm creation

    # Write your code here:
    pass

def read_note(filename):
    """Read and display a note file"""
    # Step 1: Safely read the file
    # Step 2: Display formatted content

    # Write your code here:
    pass

# Test your functions (uncomment when ready)
# create_note('output/my_note.txt', 'Learning Python', 'Today I learned file handling!')
# read_note('output/my_note.txt')
```

Challenge 2: Data Processing Pipeline (Intermediate)

Build a complete data processing pipeline:

```python
# Your task: Create a data processing pipeline
# Read sales data, process it, and generate multiple reports

import csv
import json
from collections import defaultdict

def process_sales_data():
    """Complete sales data processing pipeline"""

    # Step 1: Read sales data from CSV
    # Step 2: Calculate various statistics
    # Step 3: Generate text report
    # Step 4: Generate JSON summary
    # Step 5: Create CSV with processed data

    # Initialize data structures
    sales_data = []
    category_stats = defaultdict(lambda: {'total': 0, 'count': 0, 'items': []})

    # Write your code here:


    print("✅ Data processing pipeline completed!")
    print("📄 Generated files:")
    print("  - output/sales_report.txt")
    print("  - output/sales_summary.json")
    print("  - output/processed_sales.csv")

# Run the pipeline (uncomment when ready)
# process_sales_data()
```

Challenge 3: Log Monitoring System (Advanced)

Build a comprehensive log monitoring system:

```python
# Your task: Create a log monitoring and alerting system
# Analyze logs, detect patterns, and generate alerts

import re
from datetime import datetime, timedelta
from collections import Counter, defaultdict

class LogMonitor:
    def __init__(self, log_file):
        self.log_file = log_file
        self.alerts = []
        self.stats = defaultdict(int)

    def analyze_logs(self):
        """Analyze log file and detect issues"""
        # Step 1: Read and parse log entries
        # Step 2: Count different log levels
        # Step 3: Detect error patterns
        # Step 4: Check for time-based anomalies
        # Step 5: Generate alerts

        # Write your code here:
        pass

    def generate_report(self):
        """Generate comprehensive monitoring report"""
        # Step 1: Create summary statistics
        # Step 2: List all alerts
        # Step 3: Provide recommendations
        # Step 4: Save to file

        # Write your code here:
        pass

# Test the log monitor (uncomment when ready)
# monitor = LogMonitor('data/application.log')
# monitor.analyze_logs()
# monitor.generate_report()
```

Challenge 4: File Backup System (Expert)

Create an automated backup system with versioning:

```
# Your task: Create a comprehensive backup system
# Include versioning, compression, and integrity checks

import os
import shutil
import hashlib
from datetime import datetime
import json

class BackupSystem:
    def __init__(self, source_dir, backup_dir):
        self.source_dir = source_dir
        self.backup_dir = backup_dir
        self.backup_log = []

    def create_backup(self):
        """Create timestamped backup with integrity checks"""
        # Step 1: Create timestamped backup directory
        # Step 2: Copy files with verification
        # Step 3: Generate checksums
        # Step 4: Create backup manifest
        # Step 5: Log the backup operation

        # Write your code here:
        pass

    def verify_backup(self, backup_path):
        """Verify backup integrity"""
        # Step 1: Read backup manifest
        # Step 2: Verify file checksums
        # Step 3: Report verification results

        # Write your code here:
        pass

    def list_backups(self):
        """List all available backups"""
        # Step 1: Scan backup directory
        # Step 2: Read backup manifests
        # Step 3: Display backup information

        # Write your code here:
        pass

# Test the backup system (uncomment when ready)
# backup_system = BackupSystem('data', 'backups')
# backup_system.create_backup()
# backup_system.list_backups()
```

Final Project: Complete File Management System

Combine everything you've learned into a comprehensive file management system:

```
# Your final challenge: Create a complete file management system
# Features: file operations, data processing, monitoring, and backup

class FileManager:
    """Complete file management system"""

    def __init__(self, workspace_dir):
        self.workspace = workspace_dir
        self.ensure_workspace()

    def ensure_workspace(self):
        """Create workspace directory structure"""
        # Create necessary directories
        pass

    def process_csv_data(self, csv_file):
        """Process CSV data and generate reports"""
        # Implement CSV processing
```

```
        pass

    def monitor_logs(self, log_file):
        """Monitor log files and generate alerts"""
        # Implement log monitoring
        pass

    def backup_files(self, source_pattern):
        """Backup files matching pattern"""
        # Implement backup functionality
        pass

    def generate_dashboard(self):
        """Generate HTML dashboard with all information"""
        # Create comprehensive dashboard
        pass

# Your implementation here:
# Create an instance and test all features

print("Final Project: File Management System")
print("Implement all the features you've learned:")
print("- File reading/writing with error handling")
print("- CSV and JSON processing")
print("- Log analysis and monitoring")
print("- Backup and versioning")
print("- Dashboard generation")
print("\nGood luck! ")
```

```
Final Project: File Management System
Implement all the features you've learned:
- File reading/writing with error handling
- CSV and JSON processing
- Log analysis and monitoring
- Backup and versioning
- Dashboard generation

Good luck!
```

Conclusion and Next Steps

What You've Accomplished

Congratulations! You've completed a comprehensive journey through Python file handling. You now have:

Core Skills Mastered

- **Basic File Operations**: open, read, write, close
- **Safe File Handling**: Using `with` statements
- **Error Handling**: Graceful exception management
- **Text Encoding**: UTF-8 and international characters

Advanced Techniques

- **CSV Processing**: Reading and writing structured data
- **JSON Handling**: Configuration and API data
- **Log Analysis**: Real-world text processing
- **File Modes**: Understanding r, w, a, x modes

Professional Practices

- **Best Practices**: Production-ready code patterns
- **Error Recovery**: Robust error handling
- **Performance**: Memory-efficient file processing
- **Security**: Safe file operations

Next Steps in Your Python Journey

Immediate Applications

- **Data Analysis**: Process real datasets with pandas
- **Web Development**: Handle user uploads and configuration

- **Automation**: Create file processing scripts
- **System Administration**: Log analysis and monitoring

Advanced Topics to Explore

- **Binary Files**: Images, videos, and binary data
- **Database Integration**: SQLite and file-based databases
- **Network Files**: FTP, HTTP file operations
- **Compression**: ZIP, GZIP file handling

Recommended Libraries

- **Pandas**: Advanced data file processing
- **Pathlib**: Modern file path handling
- **Openpyxl**: Excel file manipulation
- **Requests**: Download files from web APIs

Keep Practicing!

The best way to master file handling is through practice with real data:

1. **Find real datasets** online (Kaggle, government data)
2. **Build practical projects** (log analyzers, data processors)
3. **Contribute to open source** projects using file handling
4. **Create your own tools** for daily file management tasks

Thank you for completing this tutorial!

You're now equipped with solid file handling skills that will serve you well in your Python programming journey. Remember: practice makes perfect, and real-world projects are the best teachers.

Happy Coding!