Python Programming

Unit 03 – Lecture 01: File Handling Fundamentals and Directories

Tofik Ali

School of Computer Science, UPES Dehradun

February 14, 2026

Repository: `https://github.com/tali7c/Python-Programming`

Quick Links

Core Concepts Demo Interactive Summary

Agenda

Learning Outcomes

- Open files using correct access modes (read/write/append)

Learning Outcomes

- Open files using correct access modes (read/write/append)
- Read and write text data using common file methods

Learning Outcomes

- Open files using correct access modes (read/write/append)
- Read and write text data using common file methods
- Use the `with` statement to manage file resources safely

Learning Outcomes

- Open files using correct access modes (read/write/append)
- Read and write text data using common file methods
- Use the `with` statement to manage file resources safely
- Work with directories using `pathlib` / `os`

Learning Outcomes

- Open files using correct access modes (read/write/append)
- Read and write text data using common file methods
- Use the `with` statement to manage file resources safely
- Work with directories using `pathlib` / `os`
- Apply file handling to simple real-world tasks (logs, reports, data)

Why File Handling?

- Data should **persist** after the program ends

Why File Handling?

- Data should **persist** after the program ends
- Files help in: logs, configuration, reports, datasets

Why File Handling?

- Data should **persist** after the program ends
- Files help in: logs, configuration, reports, datasets
- Working with files is a foundation for:

Why File Handling?

- Data should **persist** after the program ends
- Files help in: logs, configuration, reports, datasets
- Working with files is a foundation for:
  - CSV processing

Why File Handling?

- Data should **persist** after the program ends
- Files help in: logs, configuration, reports, datasets
- Working with files is a foundation for:
  - CSV processing
  - data analysis pipelines

Why File Handling?

- Data should **persist** after the program ends
- Files help in: logs, configuration, reports, datasets
- Working with files is a foundation for:
  - CSV processing
  - data analysis pipelines
  - web and GUI apps that store data

The Basic Pattern

```
with open("data.txt", "r", encoding="utf-8") as f:
    for line in f:
        print(line.strip())
```

- `open()` returns a file object

The Basic Pattern

```
with open("data.txt", "r", encoding="utf-8") as f:
    for line in f:
        print(line.strip())
```

- `open()` returns a file object
- Always close files (the `with` block does it automatically)

File Modes (Text)

Mode	Meaning
`r`	read (file must exist)
`w`	write (overwrite / create new)
`a`	append (write at end / create new)
`x`	exclusive create (fail if exists)
`t`	text mode (default)
`b`	binary mode (images, PDFs, etc.)
`+`	update (read + write)

Reading from a File

Common methods:

- `read()` → whole file as one string

```
with open("names.txt", "r") as f:
    for line in f:
        name = line.strip()
        print(name)
```

Reading from a File

Common methods:

- `read()` → whole file as one string
- `readline()` → one line

```
with open("names.txt", "r") as f:
    for line in f:
        name = line.strip()
        print(name)
```

Reading from a File

Common methods:

- `read()` → whole file as one string
- `readline()` → one line
- `readlines()` → list of lines

```
with open("names.txt", "r") as f:
    for line in f:
        name = line.strip()
        print(name)
```

Reading from a File

Common methods:

- `read()` → whole file as one string
- `readline()` → one line
- `readlines()` → list of lines
- Best practice: iterate line-by-line (memory friendly)

```
with open("names.txt", "r") as f:
    for line in f:
        name = line.strip()
        print(name)
```

Writing to a File

- `write()` writes a string (you manage newlines)

```
lines = ["Alice\n", "Bob\n", "Charlie\n"]
with open("names.txt", "w") as f:
    f.writelines(lines)
```

Writing to a File

- `write()` writes a string (you manage newlines)
- `writelines()` writes a list of strings

```
lines = ["Alice\n", "Bob\n", "Charlie\n"]
with open("names.txt", "w") as f:
    f.writelines(lines)
```

The `with` Statement

- A **context manager** that guarantees closing the file

The `with` Statement

- A **context manager** that guarantees closing the file
- Even if an exception happens, the file is closed properly

The `with` Statement

- A **context manager** that guarantees closing the file
- Even if an exception happens, the file is closed properly
- Reduces resource leaks and locking issues

File Pointer: `tell()` and `seek()`

- `tell()` gives current position

```
with open("data.txt", "r") as f:
    print(f.tell())
    first = f.readline()
    f.seek(0)
    again = f.readline()
```

File Pointer: `tell()` and `seek()`

- `tell()` gives current position
- `seek(pos)` moves to a position

```
with open("data.txt", "r") as f:
    print(f.tell())
    first = f.readline()
    f.seek(0)
    again = f.readline()
```

File Pointer: `tell()` and `seek()`

- `tell()` gives current position
- `seek(pos)` moves to a position
- Useful when you need to re-read or skip parts

```
with open("data.txt", "r") as f:
    print(f.tell())
    first = f.readline()
    f.seek(0)
    again = f.readline()
```

Working with Directories

- Use `pathlib.Path` for readable path handling

```python
from pathlib import Path
p = Path(".")
for f in p.iterdir():
    print(f)
```

Working with Directories

- Use `pathlib.Path` for readable path handling
- Common tasks:

```python
from pathlib import Path
p = Path(".")
for f in p.iterdir():
    print(f)
```

Working with Directories

- Use `pathlib.Path` for readable path handling
- Common tasks:
  - list files in a folder

```
from pathlib import Path
p = Path(".")
for f in p.iterdir():
    print(f)
```

Working with Directories

- Use `pathlib.Path` for readable path handling
- Common tasks:
  - list files in a folder
  - create directories

```python
from pathlib import Path
p = Path(".")
for f in p.iterdir():
    print(f)
```

Working with Directories

- Use `pathlib.Path` for readable path handling
- Common tasks:
  - list files in a folder
  - create directories
  - join paths safely

```
from pathlib import Path
p = Path(".")
for f in p.iterdir():
    print(f)
```

Demo: Read/Write + Directory Listing

- `demo/file_read_write_demo.py`

Demo: Read/Write + Directory Listing

- `demo/file_read_write_demo.py`
  - writes a small file in `data/`

Demo: Read/Write + Directory Listing

- `demo/file_read_write_demo.py`
  - writes a small file in `data/`
  - reads it back and computes simple stats

Demo: Read/Write + Directory Listing

- `demo/file_read_write_demo.py`
  - writes a small file in `data/`
  - reads it back and computes simple stats
- `demo/directory_walk_demo.py`

Demo: Read/Write + Directory Listing

- `demo/file_read_write_demo.py`
  - writes a small file in `data/`
  - reads it back and computes simple stats
- `demo/directory_walk_demo.py`
  - lists files and folders under the lecture directory

Checkpoint 1

Question: What is the difference between modes `"w"` and `"a"`?

Write one example for each.

Checkpoint 2

Question: What happens if you open a non-existing file using `"r"`?

Think-Pair-Share

Discuss:

- What should be stored in files vs kept only in memory?
- Give 2 examples for each category.

Key Takeaways

- Choose correct file mode: `r/w/a/x` (text vs binary)

Key Takeaways

- Choose correct file mode: `r/w/a/x` (text vs binary)
- Prefer `with open(...)` to auto-close files safely

Key Takeaways

- Choose correct file mode: `r/w/a/x` (text vs binary)
- Prefer `with open(...)` to auto-close files safely
- Read line-by-line for large files

Key Takeaways

- Choose correct file mode: `r/w/a/x` (text vs binary)
- Prefer `with open(...)` to auto-close files safely
- Read line-by-line for large files
- Use `pathlib` for clean directory and path handling

Exit Question

Write a code snippet to read all lines from `"names.txt"` safely using `with open(...)` and print them without newline characters.