ZyBooks Chapter 12: Files in Python

Jeremy Evert

October 20, 2025

Contents

Chapter 1

12.1 Reading Files

Overview

In this section, students learn how to read from files using Python's built-in `open()` function. Files allow programs to save data permanently and later retrieve it. Instead of entering data manually each time a program runs, we can read information directly from a file.

Learning goals:

- Understand how to open, read, and close text files in Python.
- Explore the difference between `read()`, `readline()`, and `readlines()`.
- Learn to process data from files (e.g., computing averages).

—

1.1 Reading from a File

The most basic way to read data from a file is with `open()` and `read()`.

Listing 1.1: Reading text from a file.

```
# Example 1: Reading the entire contents of a file

# Open the file in read mode
myjournal = open("journal.txt")

# Read the entire file into a single string
contents = myjournal.read()

# Display what was read
print(contents)

# Close the file after use
myjournal.close()
```

Key points:

- `open("filename")` creates a file object.
- `read()` reads all text at once and returns it as a string.
- Always close the file using `close()` when done.

—

1.2 A More Complete Example

This version adds print statements and clarifies program flow.

Listing 1.2: Creating a file object and reading text.

```
print("Opening file myfile.txt.")
f = open("myfile.txt")  # create file object

print("Reading file myfile.txt.")
contents = f.read()     # read text into a string

print("Closing file myfile.txt.")
f.close()               # close the file

print("\nContents of myfile.txt:")
print(contents)
```

Tip: The file must be in the same directory as your Python script unless you specify a full path (e.g., `C:\Users\evertj\myfile.txt`).

—

1.3 Reading Line by Line

The `readlines()` method reads each line into a list of strings.

Listing 1.3: Reading all lines into a list.

```
# Example 2: Read lines from a file
my_file = open("readme.txt")
lines = my_file.readlines()

# Print the second line (remember, Python starts counting at 0)
print(lines[1])

my_file.close()
```

Note: Each element of `lines` includes the newline character `"\n"`.

—

1.4 Processing Data from a File

Programs often read data from files to compute a result, such as an average.

Listing 1.4: Calculating the average value of integers stored in a file.

```
# Example 3: Calculating an average from a file

print("Reading in data...")
f = open("mydata.txt")
lines = f.readlines()
f.close()

# Process data
print("\nCalculating average...")
total = 0
for ln in lines:
    total += int(ln)

avg = total / len(lines)
print(f"Average value: {avg}")
```

This example demonstrates:

- How to iterate through file lines.
- Converting strings to integers using `int()`.
- Computing an average from numeric data.

—

1.5 Iterating Directly Over a File Object

Python lets you loop through a file directly, one line at a time.

Listing 1.5: Iterating over the lines of a file.

```
"""Echo the contents of a file."""
f = open("myfile.txt")

for line in f:
    print(line, end="")  # end="" avoids double newlines

f.close()
```

This approach is memory-efficient and ideal for large files.

—

1.6 Practice Exercise

Challenge: Create a Python program that reads a filename from user input, opens that file, and prints its contents in uppercase.

Listing 1.6: Challenge Activity: Read and modify file contents.

```
# Example 4: Read and transform file content
```

```
filename = input("Enter filename: ")

with open(filename) as f:        # 'with' auto-closes the file
    contents = f.read()

print(contents.upper())
```

—

1.7 Explore More

For additional reading and examples:

- Python Documentation: Reading and Writing Files
- W3Schools: Python File Handling
- Real Python: Working with Files in Python

—

Summary

- Use `open()` to access a file.
- `read()`, `readline()`, and `readlines()` offer flexibility.
- Always close files, or use the `with` statement.
- Practice reading, processing, and displaying file data.

Chapter 2

12.2 Writing Files

Overview

Programs write to files to store data permanently. The `file.write()` method writes a string argument to a file. This section teaches how to open files for writing, the difference between modes, and why errors occur when you write the wrong data type.

Helpful documentation:

- Python Docs: Reading and Writing Files
- `open()` built-in function
- `io.TextIOBase.write()`
- `os.fsync()`

2.1 Writing to a File

Listing 2.1: Basic example: Writing text to a file.

```
def write_basic_example():
    """Write two lines to a new file."""
    with open("myfile.txt", "w", encoding="utf-8") as f:
        f.write("Example string.\n")
        f.write("test....\n")
    print("File written successfully!")

write_basic_example()
```

Key idea: Opening a file in mode `"w"` creates it if missing and overwrites it if it already exists.

—

2.2 Why Some Writes Fail

Listing 2.2: Example: Handling write errors gracefully.

```
def demonstrate_wrong_write():
    """Show why writing numbers directly causes a TypeError."""
    try:
        with open("wrong_write.txt", "w", encoding="utf-8") as f:
            # This will fail: write() only accepts strings.
            f.write(10.0)
    except TypeError as e:
        print("Caught error:", e)
        print("Explanation: write() in text mode needs a string,
           not a number.")

    # Correct way: convert numbers to strings.
    with open("right_write.txt", "w", encoding="utf-8") as f:
        num1, num2 = 5, 7.5
        total = num1 + num2
        f.write(str(num1))
        f.write(" + ")
        f.write(str(num2))
        f.write(" = ")
        f.write(str(total))
        f.write("\n")
    print("Fixed version works correctly!")

demonstrate_wrong_write()
```

—

2.3 File Modes

Mode	Description	Read?	Write?	Overwrite?
r	Read only	Yes	No	No
w	Write (overwrite)	No	Yes	Yes
a	Append to end	No	Yes	No
r+	Read and write (must exist)	Yes	Yes	No
w+	Read and write (truncates)	Yes	Yes	Yes
a+	Read and append	Yes	Yes	No
x	Create new file (error if exists)	No	Yes	N/A

Listing 2.3: Example: Trying wrong modes, then fixing.

```
from pathlib import Path
import io

def show_file(path):
    p = Path(path)
    print(f"\n[{p.name}] contents:")
    print(p.read_text(encoding="utf-8") if p.exists() else "<
       missing>")
```

```
def demonstrate_modes():
    path = "modes_demo.txt"
    Path(path).write_text("START\n", encoding="utf-8")

    # Wrong: open in read mode, try to write
    try:
        with open(path, "r", encoding="utf-8") as f:
            f.write("APPEND\n")
    except io.UnsupportedOperation as e:
        print("Error:", e)
        print("Explanation: 'r' is read-only; writing is not
           allowed.")

    # Correct: append mode
    with open(path, "a", encoding="utf-8") as f:
        f.write("APPEND\n")
    show_file(path)

demonstrate_modes()
```

—

2.4 Buffered Output

Python buffers output before writing to disk. This means data may not appear immediately in the file system until a newline, `flush()`, or `close()`.

Listing 2.4: Example: Forcing a buffer flush.

```
import os, time
from pathlib import Path

def buffering_demo(path="buffer_demo.txt"):
    p = Path(path)
    if p.exists():
        p.unlink()

    f = open(path, "w", encoding="utf-8")
    f.write("Write me (no newline)")  # stays in memory
    print("Before flush: file may still be empty.")
    time.sleep(1)
    f.flush()           # Push data from Python to OS
    os.fsync(f.fileno())  # Ask OS to sync to disk
    f.close()
    print("After flush: file contents written.")

buffering_demo()
```

—

2.5 Quiz Demonstrations

Listing 2.5: Mini-quiz code with real behavior.

```
def quiz_behavior():
    print("\n1) f.write(10.0) produces error:")
    try:
        with open("q1.txt", "w", encoding="utf-8") as f:
            f.write(10.0)
    except TypeError as e:
        print("True:", e)

    print("\n2) write() does NOT always write immediately:")
    with open("q2.txt", "w", encoding="utf-8") as f:
        f.write("hi")  # buffered
        print("Immediate read:", open("q2.txt").read())
        f.flush()
    print("After flush: data is saved.")

    print("\n3) flush()/fsync() forces output to disk:")
    import os
    with open("q3.txt", "w", encoding="utf-8") as f:
        f.write("sync me")
        f.flush()
        os.fsync(f.fileno())
    print("True: Data synced to disk.")

quiz_behavior()
```

—

2.6 Safe File Creation

Listing 2.6: Use 'x' mode to avoid accidental overwrite.

```
def create_once(filename="create_once.txt"):
    try:
        with open(filename, "x", encoding="utf-8") as f:
            f.write("File created successfully.\n")
        print("Created new file:", filename)
    except FileExistsError:
        print("File already exists; not overwritten.")

create_once()
create_once()
```

—

Summary

- Mode `"w"` overwrites, `"a"` appends, `"x"` creates new.
- `write()` requires strings; convert numbers with `str()` or f-strings.
- Output may be buffered; use `flush()` or close the file.
- Use `with open(...)` to ensure the file closes automatically.

Notes

This book was created to accompany the zyBooks interactive textbook, Chapter 12: Files. Each section demonstrates the concepts with well-structured LaTeX code examples and clear explanations.