**ScopeTreasure Courses**

PYTHON COURSES · LESSON PY.10 · CODE SCOPE SERIES

# Working with Files

Programs can save and read files so data survives after the program ends — learn open, read and write with the safe with-statement.

## Introduction

Everything you have stored so far lived in variables — and disappeared the moment the program ended. Run the program again, and your lists, marks and names are gone. **Files** fix that. A program can write data into a text file, and another run of the program (or another program entirely) can read it back.

Python's file work has three simple ideas: a **mode** that says what you want to do, an **open** step, and a **close** step. The mode `"w"` writes a new file (and erases an old one with the same name), `"r"` reads a file, and `"a"` appends — adds new lines at the end without erasing what is already there.

The safest way to do all of this is the **`with` statement**. It opens the file, lets you work with it, and then closes it automatically — even if something goes wrong in the middle. You will use `with` for every file in this lesson, and it should become your habit for every file in every program.

*(Roman Urdu: Variable mein rakha data program band hotay hi khatam ho jata hai — file mein likha data mehfooz rehta hai, aur dobara parha ja sakta hai.)*

File code fails in one predictable place: the gap between what you think is in the file and what is actually there. Always know which step created the file, which mode you used, and whether an earlier run already appended lines to it. When a file surprises you, open it in an editor and look — guessing about file contents wastes more time than checking.

## Key points

- **Open a file with `open()` and a mode.** `open("notes.txt", "w")` writes, `"r"` reads, `"a"` appends.
- **Always use the `with` statement.** `with open("notes.txt", "w") as f:` closes the file for you automatically when the block ends.
- **`.write()` writes text, without adding a new line by itself.** Add `"\n"` yourself when you want each entry on its own line.
- **`.read()` reads the whole file as one string.** `.readlines()` reads it as a list of lines; looping over the file reads one line at a time.
- **Mode `"w"` erases an existing file.** If the file already has data you want to keep, use `"a"` to append instead.
- **Reading a missing file gives an error.** Trying to open a file that does not exist in `"r"` mode fails — you will handle this properly with try/except in PY.11.
- **The file is created in the same folder as your program** unless you give a longer path. Check that folder if you cannot find it.

- **Close files to save your data.** The `with` statement handles this, which is why you should never skip it.
- **`.readlines()` returns a list of lines.** Each line still ends with `"\n"`, so strip before comparing or printing.
- **Reading twice gives nothing the second time.** After `.read()`, the file position is at the end; a second `.read()` returns `""`. Reopen, or read once and keep the result.

## Examples

### Example 1 — Writing lines to a file

```
with open("notes.txt", "w") as f:
    f.write("Python is fun\n")
    f.write("Today I learned files\n")

print("Notes saved")
```

What you will get on the screen:

```
Notes saved
```

And a file called `notes.txt` will appear in the same folder, containing:

```
Python is fun
Today I learned files
```

The `"\n"` at the end of each string is what puts each note on its own line. Without it, both notes would run together on one line.

### Example 2 — Reading a file back, line by line

Assume `notes.txt` from Example 1 already exists. Then run:

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

What you will get:

```
Python is fun
Today I learned files
```

`.strip()` removes the newline character at the end of each line, so `print()` does not add a second blank line. Looping over the file is the memory-friendly way to read — it handles one line at a time, even for large

files.

### Example 3 — A tiny notes app: append, then read all

```
def add_note(text):
    with open("my_notes.txt", "a") as f:
        f.write(text + "\n")

def show_notes():
    with open("my_notes.txt", "r") as f:
        print(f.read())

add_note("Buy milk")
add_note("Practise Python for 20 minutes")
show_notes()
```

What you will get the first time you run it:

```
Buy milk
Practise Python for 20 minutes
```

Run it a second time, and the file will hold four lines — because `"a"` appends instead of erasing. That is exactly how a real notes app behaves: old notes stay, new notes join them. To start fresh, delete the file, or write with `"w"` once.

### Example 4 — Reading with readlines and counting lines

Sometimes you want the whole file as a list — to count the lines, or to pick one out by position. `.readlines()` does exactly that:

```
with open("notes.txt", "w") as f:
    f.write("Python is fun\n")
    f.write("Files save data\n")
with open("notes.txt", "r") as f:
    lines = f.readlines()
print(lines)
print("Lines:", len(lines))
```

What you will get:

```
['Python is fun\n', 'Files save data\n']
Lines: 2
```

Each list item still carries its `"\n"` at the end — that is why printing the raw list shows `\n` inside the quotes. Use `.strip()` on a line before comparing it with anything, as in Example 2.

### Example 5 — A simple marks file, read back and understood

A plain text file can hold structured data if every line follows the same pattern — here, name, comma, mark. Reading it back means splitting each line on the comma:

```
with open("marks.txt", "w") as f:
    f.write("Ali,85\n")
    f.write("Ayesha,92\n")
with open("marks.txt", "r") as f:
    for line in f:
        name, mark = line.strip().split(",")
        print(f"{name} scored {mark}")
```

What you will get:

```
Ali scored 85
Ayesha scored 92
```

`line.strip().split(",")` turns `"Ali,85"` into the list `["Ali", "85"]`, and unpacking assigns both parts at once. This name-comma-value pattern is the ancestor of real CSV files, which you will meet in data work later.

### Example 6 — Real-world case: a small expense log

A household expense log is just Example 3 with numbers: append each expense as a line, then read the file and add the amounts up. Run this in a fresh folder, so the file starts empty:

```
def add_expense(item, amount):
    with open("expenses.txt", "a") as f:
        f.write(f"{item},{amount}\n")

add_expense("Tea", 200)
add_expense("Copy", 150)
total = 0
with open("expenses.txt", "r") as f:
    for line in f:
        total = total + int(line.strip().split(",")[1])
print("Total expenses:", total)
```

What you will get (in a fresh folder):

```
Total expenses: 350
```

Each line's second part is the amount as text, so `int()` converts it before adding. Run the program a second time and the total becomes 700, because the old lines are still there — append mode remembers, which is the point, but also the trap.

## Quick recap

- Files keep data after the program ends.
- Modes: `"w"` writes (erases old content), `"r"` reads, `"a"` appends.
- Always use `with open(...) as f:` so the file closes automatically.
- `.write()` needs `"\n"` for new lines; loop over the file to read line by line.
- Never use `"w"` on a file whose old content you want to keep.
- `.readlines()` gives a list of lines, each ending with a newline.
- A second `.read()` returns empty unless you reopen the file.

## Common mistakes

1. **Using `"w"` when you meant `"a"`** — your old notes vanish. Fix: use `"a"` to add, and `"w"` only when you truly want a fresh file.
2. **Forgetting `"\n"` in write** — all entries join into one long line. Fix: end every written entry with `"\n"`.
3. **Forgetting the `with` statement and never closing** — data may not be saved, and the file may stay locked. Fix: always wrap file work in `with`.
4. **Reading a file that does not exist** — `"r"` mode on a missing file gives an error. Fix: write the file first, or check it exists before reading (error handling comes in PY.11).
5. **Looking for the file in the wrong folder** — the file lands next to your program, not on your desktop or downloads folder. Fix: check the folder where your `.py` file lives.

1. **Calling `.read()` twice** — the first call reads to the end of the file, so the second returns an empty string. Fix: store the first result in a variable and reuse that variable.
2. **Writing a number without converting it** — `f.write(100)` fails, because `.write()` only accepts text. Fix: write `f.write(str(100))` or use an f-string.

## Practice

1. Write a program that saves three of your goals into `goals.txt` with `"w"`, then reads the file back and prints each line.
2. Append a fourth goal to the same file with `"a"`. Read the file again and confirm all four goals are there.
3. Build the tiny notes app from Example 3 with your own two notes. Run it twice and explain, in one line, why the second run shows four lines.
4. Try reading a file that does not exist, for example `missing.txt`, and read the error message. Write down the error name — you will learn to handle it in the next lesson.
5. Write five lines into `diary.txt`. Read it back with `.readlines()`, print the list, and print how many lines it has.
6. Save four subject marks into `subjects.txt`, one per line as `subject,mark`. Read the file back, add the marks up, and print the average.

7. Write a program that copies `notes.txt` into `notes_backup.txt`: read the first file line by line and write each line into the second. Then read the backup back and confirm it matches.

## Practice quiz

**1. Why open a file with the with statement?**

- It reads faster
- It closes the file automatically when done, even if an error happens
- It hides the file
- It makes the file bigger

**Answer:** It closes the file automatically when done, even if an error happens

**2. What does opening a file in 'w' mode do?**

- Reads it only
- Writes to it, replacing what was there before
- Appends one line
- Renames it

**Answer:** Writes to it, replacing what was there before

**3. What does readlines() give you?**

- One character
- A list with one item per line of the file
- The file's name
- The number of words

**Answer:** A list with one item per line of the file

---

**Watch the video lesson:** https://youtu.be/Qq_HpY_u8MI

**Read online:** https://scopetreasure.com/notes/py-files · More free tools and notes at https://scopetreasure.com