Writing Files in Python

Introduction

In the previous guide, we learned how to **read data from files**.

But real programs do not only read information. They also need to **save new information**.

For example:

- A notes app saves your notes.
- A student management system saves student records.
- A website saves user information.
- A logging system saves application events.
- A program may generate a report and save it as a file.

Python allows us to write data into files using the built-in `open()` function.

The basic idea is:

```
Open file
   ↓
Write data
   ↓
Close file
```

In modern Python, we normally use `with open(...)` so Python handles closing the file automatically.

---

1. What Does Writing to a File Mean?

Writing to a file means putting data into a file so that the information can be stored and used later.

For example, suppose we have:

`notes.txt`

Before writing:

After writing:

```
Learn Python
Build projects
Practice every day
```

The data remains in the file even after the Python program stops.

This is different from a normal variable.

```
notes = "Learn Python"
```

The variable exists while the program is running.

A file allows the information to be stored outside the program.

---

2. Opening a File for Writing

To write to a file, use:

```
open("filename", "w")
```

The `"w"` means:

`write`

Example:

```
file = open("notes.txt", "w")
file.write("Learn Python")
file.close()
```

This creates `notes.txt` if it does not already exist.

If the file already exists, `"w"` can replace its existing contents.

That behavior is extremely important.

---

3. The `write()` Method

The `write()` method puts text into a file.

```
file = open("notes.txt", "w")

file.write("Learn Python")

file.close()
```

The file will contain:

```
Learn Python
```

You can also write multiple pieces of text.

```
file = open("notes.txt", "w")

file.write("Learn Python")
file.write("Build projects")

file.close()
```

The result will be:

```
Learn PythonBuild projects
```

Why?

Because `write()` does not automatically add a new line.

---

4. Writing Multiple Lines

Use `\n` when you want a new line.

```
file = open("notes.txt", "w")

file.write("Learn Python\n")
file.write("Build projects\n")
file.write("Practice every day\n")

file.close()
```

The file becomes:

```
Learn Python
Build projects
```

`Practice every day`

`\n` represents a newline character.

---

5. Using `with open()`

Although this works:

```
file = open("notes.txt", "w")

file.write("Hello")

file.close()
```

a better approach is:

```
with open("notes.txt", "w") as file:
    file.write("Hello")
```

Python automatically closes the file after the `with` block finishes.

This is the recommended approach for most file operations.

---

6. Why Should We Close Files?

When Python opens a file, the operating system gives the program access to that file.

If you manually open files without closing them, resources can remain occupied.

For example:

```
file = open("notes.txt", "w")
file.write("Hello")
```

It is better to close it:

```
file.close()
```

Or use:

```
with open("notes.txt", "w") as file:
    file.write("Hello")
```

The second approach is safer because Python handles cleanup automatically.

---

7. The `w` Mode

The `"w"` mode means **write**.

Example:

```
with open("notes.txt", "w") as file:
    file.write("First version")
```

If `notes.txt` does not exist:

```
Python creates it.
```

If it already exists:

Python replaces its contents.

For example, suppose the file contains:

```
Python
JavaScript
Flutter
```

Then:

```
with open("notes.txt", "w") as file:
    file.write("Python")
```

The file becomes:

```
Python
```

The previous contents are gone.

---

8. Overwriting vs Adding

This is one of the most important concepts when working with files.

Suppose the file contains:

```
Task 1
Task 2
```

If we use:

```
with open("tasks.txt", "w") as file:
```

```
    file.write("Task 3")
```

The result is:

```
Task 3
```

The previous data was replaced.

If you want to keep the old data and add new data, you need **append mode**, which uses `"a"`.

```
with open("tasks.txt", "a") as file:
    file.write("\nTask 3")
```

Now the file becomes:

```
Task 1
Task 2
Task 3
```

Appending will be covered in detail in the next guide.

---

9. Writing Variables to a File

You can write the contents of variables.

```
name = "Hafsa"

with open("user.txt", "w") as file:
    file.write(name)
```

The file contains:

```
Hafsa
```

You can also combine text and variables.

```
name = "Hafsa"
age = 25

with open("user.txt", "w") as file:
    file.write(f"Name: {name}\n")
    file.write(f"Age: {age}")
```

The file contains:

```
Name: Hafsa
Age: 25
```

---

10. Writing Numbers

The `write()` method expects a string.

This will cause an error:

```
age = 25

with open("user.txt", "w") as file:
    file.write(age)
```

Use `str()`:

```
age = 25
```

```
with open("user.txt", "w") as file:
    file.write(str(age))
```

Or use an f-string:

```
age = 25

with open("user.txt", "w") as file:
    file.write(f"Age: {age}")
```

F-strings are usually easier when creating formatted text.

---

11. Writing a List of Strings

Python provides another useful method:

```
writelines()
```

Example:

```
lines = [
    "Python\n",
    "JavaScript\n",
    "Flutter\n"
]

with open("skills.txt", "w") as file:
    file.writelines(lines)
```

The file contains:

```
Python
JavaScript
Flutter
```

Important:

`writelines()` does **not** automatically add newline characters.

This:

```
lines = [
    "Python",
    "JavaScript",
    "Flutter"
]
```

will produce:

```
PythonJavaScriptFlutter
```

So include `\n` when you want separate lines.

---

12. Writing User Input

Files become much more useful when combined with user input.

```
name = input("Enter your name: ")

with open("user.txt", "w") as file:
    file.write(f"Name: {name}")
```

If the user enters:

```
Hafsa
```

the file contains:

```
Name: Hafsa
```

This is the beginning of building programs that can save user data.

---

13. Writing a Simple Note

We can create a basic notes program.

```
note = input("Write your note: ")

with open("notes.txt", "w") as file:
    file.write(note)

print("Note saved.")
```

The program:

1. asks the user for a note
2. opens the file
3. writes the note
4. closes the file automatically
5. tells the user the note was saved

---

14. Writing Multiple Values

Suppose we have student information:

```
name = "Hafsa"
age = 25
course = "Python"
```

We can save all of it:

```
with open("student.txt", "w") as file:
    file.write(f"Name: {name}\n")
    file.write(f"Age: {age}\n")
    file.write(f"Course: {course}\n")
```

The file becomes:

```
Name: Hafsa
Age: 25
Course: Python
```

---

15. File Encoding

When working with text files, it is good practice to specify the encoding.

```
with open("notes.txt", "w", encoding="utf-8") as file:
    file.write("Learn Python")
```

UTF-8 supports a very large range of characters.

For example:

```
with open("message.txt", "w", encoding="utf-8") as file:
    file.write("Hello دنيا")
```

Using UTF-8 helps prevent character encoding problems.

---

16. Handling Errors While Writing

File operations can fail.

For example, a program may not have permission to write to a location.

We can handle exceptions:

```
try:
    with open("notes.txt", "w", encoding="utf-8") as file:
        file.write("Hello")

except OSError:
    print("Could not write to the file.")
```

`OSError` covers many operating-system-related file problems.

You can also handle more specific exceptions when appropriate.

---

17. Writing to a File Inside a Function

File writing can be placed inside a function.

```
def save_note(note):
    with open("notes.txt", "w", encoding="utf-8") as file:
        file.write(note)

save_note("Learn Python")
```

This makes the file-writing logic reusable.

For example:

```
save_note("Practice functions")
save_note("Build a project")
```

Remember that because we are using `"w"`, each call replaces the previous content.

If we wanted to preserve previous notes, we would use append mode.

---

18. Writing a Simple Report

Suppose a program calculates a student's result.

```
name = "Hafsa"
marks = 85

with open("result.txt", "w", encoding="utf-8") as file:
    file.write("Student Result\n")
    file.write(f"Name: {name}\n")
    file.write(f"Marks: {marks}\n")
```

The generated file:

```
Student Result
Name: Hafsa
Marks: 85
```

This is an example of a program generating a report.

---

19. Writing Data From a List

Suppose we have:

```
tasks = [
    "Learn Python",
    "Practice functions",
    "Build a project"
]
```

We can write the tasks:

```
with open("tasks.txt", "w", encoding="utf-8") as file:
    for task in tasks:
        file.write(task + "\n")
```

Result:

```
Learn Python
Practice functions
Build a project
```

This pattern is common:

```
Loop through data
```

```
    ↓
Convert each item into text
    ↓
Write it to the file
```

---

20. Writing Structured Text

You can create your own simple text format.

```
users = [
    ("Hafsa", "Developer"),
    ("Asim", "Designer")
]

with open("users.txt", "w", encoding="utf-8") as file:
    for name, role in users:
        file.write(f"{name} | {role}\n")
```

The file contains:

```
Hafsa | Developer
Asim | Designer
```

Later, another program can read and process this information.

For more structured data, formats such as CSV and JSON are usually better.

---

21. Writing a haas.dev Resource List

Imagine haas.dev has a list of resources:

```
resources = [
    "Python Functions",
    "Python Lists",
    "Python Exceptions"
]
```

We can generate a simple resource file:

```
with open("resources.txt", "w", encoding="utf-8") as file:
    file.write("haas.dev Resources\n\n")

    for resource in resources:
        file.write(f"- {resource}\n")
```

The result:

```
haas.dev Resources

- Python Functions
- Python Lists
- Python Exceptions
```

This demonstrates a real-world use of file writing: generating content from program data.

---

22. Writing and Reading Work Together

File writing becomes more useful when combined with file reading.

First, write:

```
with open("message.txt", "w", encoding="utf-8") as file:
```

```
    file.write("Learn Python by building projects.")
```

Later, read:

```
with open("message.txt", "r", encoding="utf-8") as file:
    message = file.read()

print(message)
```

The overall flow is:

```
Python program
      
Generate data
      
Write to file
      
Data is stored
      
Read file later
      
Use the stored data
```

This is the foundation of persistent data storage.

---

23. Important Difference Between Variables and Files

Consider:

```
name = "Hafsa"
```

The value is stored in memory while the program is running.

When the program ends, the variable disappears.

Now consider:

```
with open("name.txt", "w") as file:
    file.write("Hafsa")
```

The information is stored in a file.

It can be accessed later by another program or another execution of the same program.

So:

```
Variable  temporary program data
File      persistent stored data
```

---

24. Common Mistakes

Mistake 1: Forgetting the `w` mode

Incorrect:

```
open("notes.txt")
```

This normally opens the file for reading.

For writing:

```
open("notes.txt", "w")
```

Mistake 2: Accidentally Overwriting Data

```
with open("notes.txt", "w") as file:
    file.write("New note")
```

If the file already contained information, it is replaced.

Always ask:

```
Do I want to replace the old data?
```

If not, append mode may be more appropriate.

Mistake 3: Forgetting Newlines

```
file.write("Python")
file.write("JavaScript")
```

Result:

```
PythonJavaScript
```

Use:

```
file.write("Python\n")
file.write("JavaScript\n")
```

Mistake 4: Writing an Integer Directly

Incorrect:

```
file.write(25)
```

Use:

```
file.write(str(25))
```

or:

```
file.write(f"{25}")
```

Mistake 5: Manually Opening Without Closing

Avoid unnecessarily writing:

```
file = open("notes.txt", "w")
file.write("Hello")
```

Prefer:

```
with open("notes.txt", "w") as file:
    file.write("Hello")
```

---

25. write() vs writelines()

Method	Purpose
write()	Writes a string

`writelines()`	Writes multiple strings
`write()`	Does not automatically add newline
`writelines()`	Does not automatically add newline

Example:

```
file.write("Python\n")
```

Multiple strings:

```
file.writelines([
    "Python\n",
    "JavaScript\n",
    "Flutter\n"
])
```

---

26. A Useful Mental Model

Think of a file as a notebook.

Opening the notebook:

```
open(...)
```

Writing something:

`write(...)`

Adding multiple lines:

`writelines(...)`

Closing the notebook:

`close()`

Using `with` means Python handles the closing step for you.

```
with open("notes.txt", "w") as file:
    file.write("Python")
```

Think:

`Open → Work → Automatically Close`

---

27. Best Practices

1. Prefer `with open()`

```
with open("data.txt", "w", encoding="utf-8") as file:
    file.write("Hello")
```

2. Specify UTF-8 for text files

`encoding="utf-8"`

3. Be careful with `"w"`

Remember:

`w` = replace existing content

4. Add newlines intentionally

```
file.write("Line 1\n")
```

5. Keep file operations focused

Instead of spreading file logic everywhere:

```
def save_report(report):
    with open("report.txt", "w", encoding="utf-8") as file:
        file.write(report)
```

6. Handle possible file errors

```
try:
    with open("report.txt", "w", encoding="utf-8") as file:
        file.write("Report")
except OSError:
    print("Could not save report.")
```

---

28. A Practical Problem-Solving Framework

When your program needs to save information, ask:

Step 1: What data needs to be saved?

Example:

```
username
email
score
```

Step 2: Should old data be replaced or preserved?

Replace → w
Preserve/add → a

Step 3: What format should the data use?

Simple text:

Name: Hafsa
Score: 90

Structured data may later use:

CSV
JSON

Step 4: Where should the file be stored?

For example:

data/
  users.txt

Step 5: How will the program handle errors?

Consider:

try:
  ...
except OSError:
  ...

This turns file writing from a simple command into a deliberate program-design decision.

---

29. Mini Project: Personal Notes Saver

Create a small notes application.

```
note = input("Enter your note: ")

with open("notes.txt", "w", encoding="utf-8") as file:
    file.write(note)

print("Your note has been saved.")
```

How it works

User enters note
↓
Python receives input
↓
File opens in write mode
↓
Note is written
↓
File closes automatically
↓
Saved message appears

Example

Input:

```
Practice Python functions today.
```

File:

Practice Python functions today.

---

30. Mini Project: Student Report Generator

```
name = input("Student name: ")
marks = input("Marks: ")

report = (
    "Student Report\n"
    f"Name: {name}\n"
    f"Marks: {marks}\n"
)

with open("student_report.txt", "w", encoding="utf-8") as file:
    file.write(report)

print("Report generated successfully.")
```

This project combines:

- variables
- input
- strings
- f-strings
- file writing
- with
- UTF-8 encoding

---

31. 5 Minute Challenge

Create a program that asks for:

```
Name
Favorite language
Current project
```

Then save the information to:

```
developer.txt
```

Expected output:

```
Developer Profile
Name: Hafsa
Favorite Language: Python
Current Project: haas.dev
```

Extra challenge

Add today's goal:

```
Today's Goal: Build a Python project
```

---

32. Exercises

Exercise 1

Create a file called:

```
welcome.txt
```

and write:

`Welcome to Python.`

Exercise 2

Write three programming languages on separate lines.

Exercise 3

Ask the user for their name and save it to a file.

Exercise 4

Create a student report containing:

```
Name
Age
Class
Marks
```

Exercise 5

Create a list of five tasks and save each task on a separate line.

Exercise 6

Create a function:

```
save_message(message)
```

that saves a message to a file.

Exercise 7

Create a haas.dev-style resource generator that receives a list of resource titles and saves them to a text file.

---

33. Mini Quiz

Question 1

Which mode is used to write to a file?

A. `"r"`

B. `"w"`

C. `"x"`

D. `"read"`

Answer: B

Question 2

What happens when `"w"` is used on an existing file?

A. The file is deleted permanently

B. The old contents can be replaced

C. Nothing happens

D. Python reads the file

Answer: B

Question 3

What does `\n` represent?

A. Space

B. Tab

C. New line

D. File ending

Answer: C

Question 4

Why is `with open()` recommended?

A. It makes files smaller

B. It automatically handles closing the file

C. It prevents all errors

D. It converts files to Python objects

Answer: B

Question 5

What does `write()` expect?

A. A string

B. Only an integer

C. Only a list

D. A dictionary only

Answer: A

---

34. Interview Questions

1. What is the purpose of the `open()` function?

It opens a file so that a Python program can read from or write to it.

2. What does `"w"` mean?

It opens a file in write mode. If the file already exists, its contents can be overwritten.

3. What is the difference between `write()` and `writelines()`?

`write()` writes a string, while `writelines()` writes multiple strings.

4. Does `write()` automatically add a newline?

No.

5. Why should `with open()` usually be preferred?

It ensures the file is properly closed after the block finishes.

6. Why might `encoding="utf-8"` be used?

To handle text consistently across a wide range of characters.

7. Can you write an integer directly using `write()`?

No. Convert it to a string or use an f-string.

8. What is the major risk of `"w"` mode?

It can replace existing file contents.

---

35. Cheat Sheet

Open for writing

```
with open("file.txt", "w") as file:
    ...
```

Write text

```
file.write("Hello")
```

Write a newline

```
file.write("\n")
```

Write multiple lines

```
file.write("Line 1\n")
file.write("Line 2\n")
```

Write a variable

```
name = "Hafsa"

file.write(name)
```

Write a number

```
age = 25

file.write(str(age))
```

Use an f-string

```
file.write(f"Age: {age}")
```

Write multiple strings

```
file.writelines([
    "Python\n",
    "JavaScript\n"
])
```

Recommended text-file pattern

```
with open("data.txt", "w", encoding="utf-8") as file:
    file.write("Hello")
```

---

36. Key Takeaways

- Python can save information to files.
- `open()` is used to open a file.
- `"w"` opens a file for writing.
- `"w"` can replace existing contents.
- `write()` writes a string.
- `writelines()` writes multiple strings.
- `write()` does not automatically add newlines.
- Use `\n` for separate lines.
- Use `with open()` so the file is automatically closed.
- `encoding="utf-8"` is a good choice for text files.
- Numbers need to be converted to strings before using `write()`, unless they are included through an f-string.
- File writing is one of the first steps toward persistent data storage.
- Later, you can combine file writing with CSV, JSON, databases, and application storage.

The core pattern to remember is:

```
with open("file.txt", "w", encoding="utf-8") as file:
    file.write("Your data here")
```

Once you understand this pattern, Python programs can move beyond temporary variables and start **saving real information for later use**.

Visit haas.dev for more resources and guides.

https://dev-roast-app.vercel.app

haas.dev
https://dev-roast-app.vercel.app/resources