

# 08. Introduction to File Handling

September 16, 2025

## 1 8. Introduction to File Handling

### 1.0.1 8.1 Opening a File

You can use the `open()` function to open a file. It takes two arguments: the file path and the mode in which you want to open the file (`r` for reading, `w` for writing, `a` for appending, and `r+` for reading and writing). For example:

```
[4]: f = open("file_example_in.txt", "r")
```

### 1.0.2 8.2 Reading from a File

For reading lines from a file, you can loop over the file object. This is memory efficient, fast, and leads to simple code:

```
[5]: for line in f:  
    print(line, end='')
```

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.
```

Another strategy is reading the entire content of the file in a variable:

```
[6]: f = open("file_example_in.txt", "r")  
content = f.read()  
print(content)
```

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.
```

### 1.0.3 8.3 Writing to a File

In order to write to a file, it should be open in write mode, for example: - `w` (write) - the file will be created if it doesn't exist and overwritten if it does exist - `a` (append) - the file will be created if it doesn't exist and anything you write to it will be added at the end

```
[7]: f = open("file_example_out.txt", "w")
```

After opening a file for writing, you can use the `write()` method to write data to the file.

```
[8]: f.write("Readability counts.")
```

```
[8]: 19
```

### 1.0.4 8.4 Closing the File

After you finish working with the file, you should call `close()` to close the file and immediately free up any system resources used by it. If you don't explicitly close a file, Python's garbage collector will eventually destroy the object and close the open file for you, but the file may stay open for a while.

```
[9]: f.close()
```

### 1.0.5 Exercises 8.4

1. Write a program that reads the content of a text file and counts the number of words in it. Print the word count.
2. Write a function `grep` that receives `text` and `file` as parameters and returns a list with all the lines in the file containing `text`.
3. Write another function `grepinto` that receives `text`, `infile` and `outfile` as parameters and writes to `outfile` the lines in `infile` that contain `text`.

[!] `file`, `infile` and `outfile` are all file names - not file handlers.

### 1.0.6 8.5 Context Managers

A context manager is an object that is used to manage resources, such as files, network connections, or database connections. Context managers are typically used with the `with` statement to ensure that resources are properly managed and released when they are no longer needed, even if an error occurs.

The built-in `open()` function returns a file object that can be used as a context manager. When used with the `with` statement, it automatically handles opening and closing the file.

```
[10]: with open("file_example_in.txt", "r") as f:  
    content = f.read()  
    # Do something with the file content
```

In this example:

- `open("file_example_in.txt", "r")` opens the file "example.txt" for reading.

- `as f` assigns the file object to the variable `f`.
- The `with` statement ensures that the file is properly closed when the block inside the `with` statement is exited, even if an exception occurs.
- Inside the block, you can perform file operations like reading, writing, or appending.

The benefit of using a context manager with files is that it ensures that resources are released promptly and reliably. It also makes the code cleaner and more readable by clearly indicating the scope in which the file is being used.

### 1.0.7 Exercises 8.5

1. Rewrite `grep` and `grepinto` functions to use context managers. For `grepinto`, note that you can open two files in a single `with` statement: “`python with open(read_file, "r") as fr, open(write_file, "w") as fw: # do something with the files`