Working with Files in Python

In this course, you'll learn how to:

- Retrieve file properties
- Create directories
- Match patterns in filenames
- Traverse directory trees
- Make temporary files and directories
- Delete files and directories
- Copy, move, or rename files and directories
- Create and extract ZIP and TAR archives
- Open multiple files using the fileinput module

TABLE OF CONTENTS

The open function

- `open(file_name, mode)` opens the given filename (from the current directory) in the specified mode

Basic File Modes

“w”	Write mode: wipes existing file content
“r”	Read mode: read-only access to the file content
“a”	Append mode: writes to the end of the file

Sample Directory

```
test/
│
├── (empty)
```

TABLE OF CONTENTS

Basic directory listing functions

With the `os` module:

- `os.listdir(dirname)` takes in a directory name as a string and returns a `list` of all of the files and subdirectories in that directory
- `os.scandir(dirname)` similar behavior to `listdir`, but returns an iterator of file objects rather than a list of strings

With the `pathlib` module:

- `pathlib.Path.iterdir()` works on a path object and returns a similar iterator to `scandir`

Sample Directory

```
my_directory/
│
├── sub_dir/
│   ├── bar.py
│   └── foo.py
│
├── sub_dir_b/
│   └── file4.txt
│
├── file1.py
├── file2.csv
└── file3.txt
```

TABLE OF CONTENTS

Useful functions for getting file information

With the `os` module:

- `os.stat(path_string)` takes in a file or directory path as a string and returns a `stat_result` object with the file data
- `os.scandir(dirname)` — In the returned iterator, each object has a `.stat()` method that returns the same data as `os.stat()`

With the `pathlib` module:

- `pathlib.Path.iterdir()` — each item in the iterator has a `.stat()` method, just like in `scandir`

Sample Directory

```
my_directory/
│
├── sub_dir/
│   ├── bar.py
│   └── foo.py
│
├── sub_dir_b/
│   └── file4.txt
│
├── file1.py
├── file2.csv
└── file3.txt
```

TABLE OF CONTENTS

Useful functions for making directories

With the `os` module:

- `os.mkdir(dir_name)` creates a single subdirectory with the given name
- `os.makedirs(path_name)` creates full directory trees (including intermediate directories, as needed)

With the `pathlib` module:

- `pathlib.Path.mkdir()` creates a directory from the given `Path` object

TABLE OF CONTENTS

Useful functions for deleting files

With the `os` module:

- `os.remove(file_path)` deletes a single file, raises `FileNotFound` if the file doesn't exist
- `os.unlink(file_path)` is essentially identical to `os.remove`

With the `pathlib` module:

- `pathlib.Path.unlink()` is identical to the `os` module options, except it operates on a `Path` object

Sample Directory

```
.
|
├── folder_1/
|   ├── file1.py
|   ├── file2.py
|   └── file3.py
|
├── folder_2/
|   ├── file4.py
|   ├── file5.py
|   └── file6.py
|
├── test1.txt
└── test2.txt
```

Useful functions for deleting directories

Deleting a single directory:

- `os.rmdir(dir_path)` deletes a single directory, raises `OSError` if the directory is non-empty
- `pathlib.Path.rmdir()` is identical to `os.rmdir()`, except it operates on a `Path` object

Deleting an entire directory tree:

- `shutil.rmtree(dir_path)` deletes the entire directory tree rooted at `dir_path`

Sample Directory

```
.
|
├── folder_1/
|   ├── file1.py
|   ├── file2.py
|   └── file3.py
|
├── folder_2/
|   ├── file4.py
|   ├── file5.py
|   └── file6.py
|
├── test1.txt
└── test2.txt
```

TABLE OF CONTENTS

Useful functions for filename matching

- `.startswith()` and `.endswith()` both operate on strings and can be useful when dealing strictly with filenames
- `fnmatch.fnmatch(file_name, unix_pattern)` takes in a filename and a Unix-style pattern string and returns whether the filename matches that pattern
- `glob.glob(pattern)` takes in a search pattern and returns a list of the files in the current directory matching that pattern
- `pathlib.Path.glob(pattern)` works just like `glob`, but operates on a `Path` object

Sample Directory

```
.
│
├── sub_dir/
│   ├── file1.py
│   └── file2.py
│
├── admin.py
├── data_01_backup.txt
├── data_01.txt
├── data_02_backup.txt
├── data_02.txt
├── data_03_backup.txt
├── data_03.txt
└── tests.py
```

TABLE OF CONTENTS

Useful functions for directory traversal

- `os.walk(dirpath, topdown=True)` takes in a directory name and returns an iterator of `(dirpath, dirnames, files)` on each iteration

Sample Directory

```
.
|
├── folder_1/
|   ├── file1.py
|   ├── file2.py
|   └── file3.py
|
├── folder_2/
|   ├── file4.py
|   ├── file5.py
|   └── file6.py
|
├── test1.txt
└── test2.txt
```

TABLE OF CONTENTS

Temporary file constructors

- `tempfile.TemporaryFile(mode)` creates and opens a temporary file in the specified mode
- `tempfile.TemporaryDirectory()` creates a temporary directory and returns it for use

TABLE OF CONTENTS

Useful functions for copying and moving files

Copying:

- `shutil.copy(src, dst)` copies a file (but not its metadata) from the `src` path to the `dst` path — if you need the metadata, use `copy2`
- `shutil.copytree(src_dir, dst_dir)` copies full directory trees

Moving/Renaming:

- `shutil.move(src, dst)` moves a file or directory from `src` to `dst`
- `os.rename(old, new)` renames a file or directory

Sample Directory

```
.
|
├── folder_1/
|   ├── file1.py
|   ├── file2.py
|   └── file3.py
|
├── folder_2/
|   ├── file4.py
|   ├── file5.py
|   └── file6.py
|
├── test1.txt
└── test2.txt
```

TABLE OF CONTENTS

Useful archive functions

- `zipfile.Zipfile(zip_name, mode)` opens a zipfile in the desired mode — works a lot like opening any other kind of file
- `zipfile.Zipfile().extract(filename)` extracts the given file from the zipfile (also see `extract_all`)
- `shutil.make_archive(basename, format, root_dir)` makes an archive with the given parameters
- `shutil.unpack_archive(archive_name, extract_dir)` extracts the given archive into the given directory

Sample Directory

```
.
|
├── sub_dir/
|   ├── bar.py
|   └── foo.py
|
|
├── file1.py
├── file2.py
└── file3.py
```

TABLE OF CONTENTS

The `fileinput` module

- `fileinput.input([filenames])` takes all of the filenames from the `filenames` list (or `sys.argv` if no list is given) and makes them into one input stream that you can operate on. It also provides some information about each line in the output, like the line number, whether it's the first line in its file, etc.

Working with Files in Python

In this course, you've learned how to:

- Retrieve file properties
- Create directories
- Match patterns in filenames
- Traverse directory trees
- Make temporary files and directories
- Delete files and directories
- Copy, move, or rename files and directories
- Create and extract ZIP and TAR archives
- Open multiple files using the fileinput module