

10 – File I/O

Bálint Aradi

Scientific Programming in Python (2026)

<https://atticlectures.net/scipro/python-2026/>

File I/O workflow

- Open file
 - Do read/write operations
 - Close file
-
- Closing of a file is optional (although recommended)
 - By using context manager blocks **with ... as ...** this can be automated
 - File will be closed upon exiting the block

```
fd = open("test.txt", "r")  
txt = fd.read()  
fd.close()
```

```
with open("test.txt", "r") as fd:  
    txt = fd.read()  
print("File closed automatically")
```

Opening a file

- The file is opened with the **open()** function:

```
fd = open(filename, mode, encoding=None)
```

File descriptor Name/path of the file to open File opening mode Encoding (for text files only)

File opening modes

r	read only
w	write only (existing file overwritten!)
a	append (data appended to the end)
r+	read/write
rb, wb, ab, r+b	same operations but for a binary file (default: text)

Encoding (only for textfiles)

- Default encoding is **system dependent**
- Always **specify "utf-8" encoding explicitly** (unless you have good reasons not to)

```
fd = open(  
    "myfile.txt",  
    mode="r",  
    encoding="utf-8"  
)
```

Reading text from a file

```
with open("test.txt", "r", encoding="utf-8") as fd:  
    ...
```

- **Iterating** over file handler returns the lines in the file as strings (including the newline character at the line ends):
- The **readlines()** method returns a list of the lines in the file:
- The **readline()** method returns the next line in the file (and empty string if all lines had been read):
- The **read()** method returns the entire file content as one string:

```
for line in fd:  
    print(line)
```

```
lines = fd.readlines()  
print(lines)
```

```
line = fd.readline()  
while line:  
    print(line)  
    line = fd.readline()
```

```
txt = fd.read()  
print(txt)
```

Writing text to a file

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

- The **write()** method writes a given string into a file
- The **writelines()** method writes a list of strings into a file

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

```
lines = ["Line1\n", "Line2\n"]  
fd.writelines(lines)
```

equiv.

```
lines = ["Line1\n", "Line2\n"]  
for line in lines:  
    fd.write(line)
```

equiv.

```
lines = ["Line1", "Line2"]  
fd.write("\n".join(lines))
```

Reading / writing arrays

- Numpy/Scipy have special routines to read/write data arrays in text form (and also in other formats)

`numpy.loadtxt()`

Reads data from a file into an array

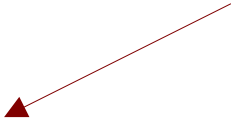
`numpy.savetxt()`

Writes array data into a file

```
data = np.loadtxt("test.dat")
data
```


```
array([[ 1.,  2.],
       [ 3.,  4.]])
```

```
test.dat:
# Some comment
1 2
3 4
```



```
data2 = np.array([1, 2, 3])
np.savetxt("test2.dat", data2)
```

```
test2.dat
1.000000000000000000000000e+00
2.000000000000000000000000e+00
3.000000000000000000000000e+00
```



Path manipulation (os.path)

os.path module

- Module with very helpful functions for file name and path manipulations
- **os.path.join()**: Joining path names:

```
import os.path
```

```
directory = "harmonic"
```

```
fname = "energies.dat"
```

```
fname_full = os.path.join(directory, fname)
```

```
fname_full
```

```
'harmonic/energies.dat'
```

See also: [os.path module documentation](#)

Path manipulation (pathlib)

pathlib module

- Object oriented path handling
- Path object offers methods and overridden operators to query and manipulate paths

```
from pathlib import Path
```

```
directory = Path("dir1/dir2")
```

```
fname = "data.dat"
```

```
fname_full = directory / fname
```

```
PosixPath('dir1/dir2')      Path-object
```

```
'data.dat'                  String
```

```
PosixPath('dir1/dir2/data.dat')
```

- Path object can be used in the **open()** statement instead of string file name

```
file = Path("test.dat")  
with open(file, "r", ...) as fd:  
    fd.read()
```

See also: [pathlib module documentation](#)