EINFÜHRUNG IN DIE PROGRAMMIERUNG

I/O

DHBW MANNHEIM
WIRTSCHAFTSINFORMATIK (DATA SCIENCE)

Markus Menth
Martin Gropp

READING & WRITING TEXT FILES

READING TEXT FILES

EXAMPLE

```
with open('file.txt') as f:
    for line in f:
        print(line, end='')
```

- open opens the file (for reading).
- The with statement is used to make sure the file is closed at the end of the block. Inside the block, the file can be accessed through the variable `f`.
- The `for` loop goes through the file line by line.
  ⚠ `line` includes the `\n` or `\r\n` characters at the end of a line!

READING TEXT FILES

ALTERNATIVE

```
with open('file.txt') as my_file:
    lines = my_file.readlines()
```

WRITING TEXT FILES

`open` takes an optional argument that determines the file *mode*:

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open a disk file for updating (reading and writing)

WRITING TEXT FILES

```
with open('file.txt', 'w') as f:
    f.write('line 1\n')
    f.write('line 2\n')
```

```
with open('file.txt', 'w') as f:
    print('line 1', file=f)
    print('line 2', file=f)
```

FURTHER READING 🔬

- binary files (i.e. not plain text)
- character encoding of text files (e.g. utf-8, utf-16, iso-8859-15, windows-1252, ...)
- `f.read`, `f.readline`

READING AND WRITING STRUCTURED DATA

JSON

JSON is a common format for exchanging *structured data* (e.g. lists, dicts, ...).

```
import json

d = {
    'a': 'foo',
    'b': [1, 2, 3],
    'c': {
        'd': 42
    }
}

print(json.dumps(d))
```

```
{"a": "foo", "b": [1, 2, 3], "c": {"d": 42}}
```

JSON

JSON can be directly written to and read from files.

```
import json

with open('input.json') as f:
    x = json.load(f)

with open('output.json', 'w') as f:
    json.dump(x, f)
```

FURTHER READING 🔬

There are many alternatives for persisting structured data:

- pickle: for general Python objects
- configparser: .ini-style config files
- csv: Comma-Separated Values
- yaml: a popular JSON alternative

PATHS

PATHS

Traditionally, file system paths are represented as strings:
`/etc/passwd` or `C:\Windows\system32`

Formats differ between operating systems.
Special care is required when working with path strings (os.path)! 🔬

```
import os.path

directory = '/var/log'
file = os.path.join(directory, 'system.log')
print(file)
```

```
/var/log/system.log
```

PATHLIB

The pathlib module has a powerful alternative:

```
from pathlib import Path

directory = Path('/var/log')
file = directory / 'system.log'
```

PATHLIB

`Path` objects can be used with `open`:

```
from pathlib import Path

path = Path.home() / '.myapp.config'

with open(path) as f:
    print(f.readlines())
```

PATHLIB

Some useful methods:

- `p.exists()`
- `p.is_file()`, `p.is_dir()`
- `p.name`
- `p.parent`
- `p.suffix`

Getting the directory containing the current Python file:

```
base_dir = Path(__file__).parent
```

LISTING DIRECTORIES

Paths can be used to list the files and sub-directories inside a directory:

```
home = Path.home()
for path in home.iterdir():
    if path.is_dir():
        print('D', str(path))
    else:
        print(' ', str(path))
```

FILE FILTERS

Directory contents can be filtered with `glob`.

EXAMPLES

Directory `d` contains the following files: `1.gif`, `2.txt`, `foo.png`.

```
>>> d = Path('d')
>>> list(d.glob('*.png'))
[PosixPath('foo.png'), PosixPath('1.png')]

>>> list(d.glob('?.png'))
[PosixPath('1.png')]

>>> list(d.glob('[0-9].*'))
[PosixPath('2.txt'), PosixPath('1.txt')]
```

(See also: glob)

TEMPORARY FILES

TEMPORARY FILES

tempfile: Create temporary files and directories, with automatic cleanup.

```
import tempfile
```

```
with tempfile.NamedTemporaryFile(mode='w') as f:
    print('Now writing to file', f.name)
    f.write('Hello world!')
```

```
f = tempfile.NamedTemporaryFile(mode='w')
f.write('Hello world!')
f.close()
```