INF1-IP

Introduction to Programming

Lecture 10 – I/O 1: Reading and writing files

Matteo Degiacomi

THE UNIVERSITY of EDINBURGH
informatics

> Today: programs that remember

- Every program so far *forgets everything* when it stops.
- Files *survive*: switch off the laptop, come back next week

Today, we write data to files and read it back:

- Escape characters: the characters you cannot see
- Paths: where files live
- Opening, reading, writing and closing files
- Parsing a file



Escape sequences

```
s = "Matteo said: \"really???\"\nFiona replied: \"yes, it\'s 42!\"\n"
```

A backslash starts an **escape sequence**: two characters in code, *one* in the string.

In your code	\n	\t	\"	\'	\\
In the string	newline	tab	double quote	single quote	backslash

Predict: how many lines does the first `print()` show? What is `len("\n")`?

```
print("Name\tScore")
print("First line\nSecond line")
print("A backslash: \\")
print("She said \"hello\".")
print('It\'s Python!')
print(len("\n"))
```

Output

```
Name    Score
First line
Second line
A backslash: \
She said "hello".
It's Python!
1
```

Making the invisible visible

`print()` *uses* the newline. The built-in `repr()` *shows* it, quotes included.

Predict: which of the two outputs fits on one line?

```
message = "tea\tsoup\ncake"
print(message)
print(repr(message))
```

Output

```
tea   soup
cake
'tea\tsoup\ncake'
```

From now on, `repr()` is our X-ray for strings: files are full of newlines we cannot see.

Try it yourself #1: a two-line timetable

Make **one** string that prints as:

```
Monday      10:00
Tuesday     11:00
```

Then print its `repr()`.



About 2 minutes · on your own

Example solution #1

```
timetable = "Monday\t10:00\nTuesday\t11:00"
print(timetable)
print(repr(timetable))
```

Output

```
Monday      10:00
Tuesday     11:00
'Monday\t10:00\nTuesday\t11:00'
```

One newline in the string: `print()` adds the final line ending itself.



> Paths: where files live

current working directory: where your program is currently operating in by default.

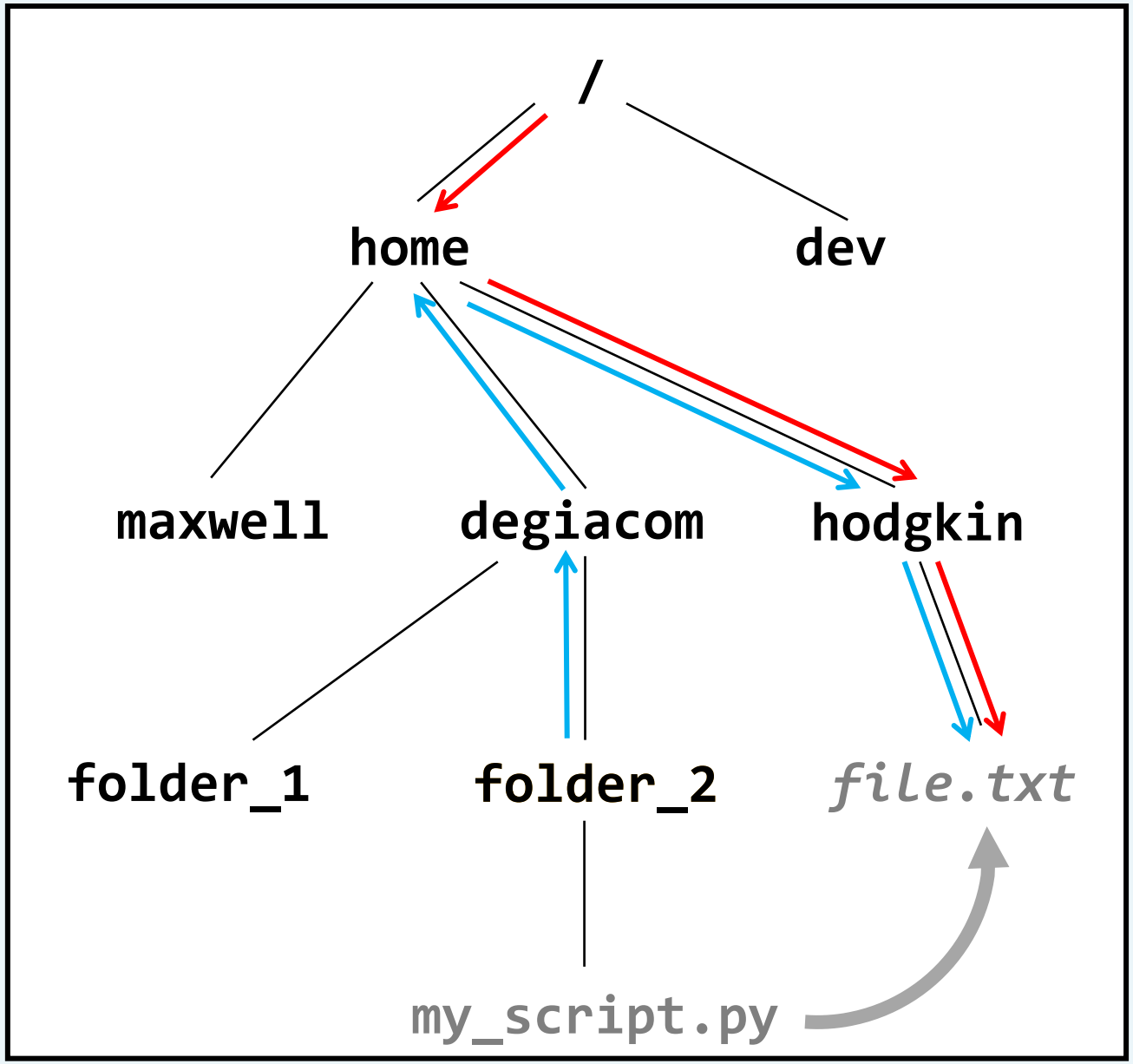
Absolute path: path to the file from file system root, e.g.:
`/home/hodgkin/input_file.txt`

Relative path: path to the file with respect to the current working directory, e.g.:
`../../hodgkin/input_file.txt`

Special path components:

`.` (one dot) means “this folder”

`..` (two dots) means “the parent folder”



! Windows paths: watch the backslash

System	Separator	Absolute path
Linux and macOS	`/`	`/home/student/data/notes.txt`
Windows	`\`	`C:\Users\student\data\notes.txt`

Predict: what does this print?

```
print("C:\new\table.txt")
```

Output

```
C:
ew	able.txt
```

- `\n` became a newline and `\t` a tab: the path changed *silently*
- `"C:\Users\student"` does not even run: `\U` starts another escape
- In Python, write Windows paths with `/`: Windows accepts it

Live coding: open, write, close

```
sample_path = "ip26_l10_sample.txt"

file = open(sample_path, "w", encoding="utf-8")
file.write("First line\nSecond line\nCafé: £2.50\n")
file.close()

print("Wrote to", sample_path)
```

- `open()` returns a **file object**, here `file`
- `"w"` means we open the file in **write mode**
- `encoding="utf-8"` stores any character, even `é` and `£`
- `.write()` does *not* add newlines for us, unlike `print()`
- `file.close()` when you have finished with the file

Watch out: `"w"` empties an existing file the moment it is opened, even if you never write to it.

Note: From Python 3.15 (2026), UTF-8 is the default. Noteable runs an older Python, so, if you have special characters, use `encoding="utf-8"`.

Reading a whole file

```
file = open(sample_path, "r", encoding="utf-8")
contents = file.read()
file.close()

print(contents)
print(type(contents))
print(repr(contents))
```

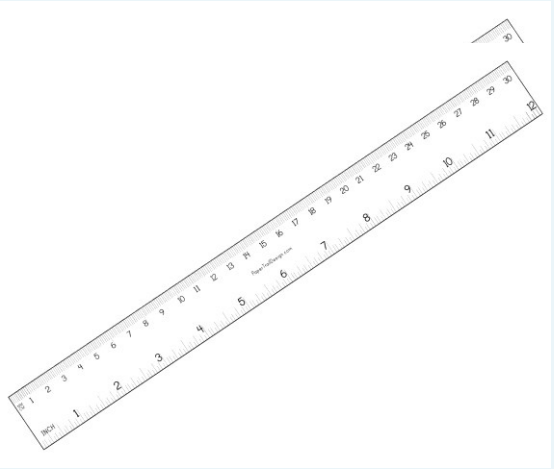
Output

```
First line
Second line
Café: £2.50

<class 'str'>
'First line\nSecond line\nCafé: £2.50\n'
```

- `"r"` is for **read**
- `.read()` returns the whole file as **one string**
- A second `.read()` on the same open file returns `''`: the file remembers how far you have read
- A missing file, or a typo in its name, raises `FileNotFoundError`

Try it yourself #2: read and measure



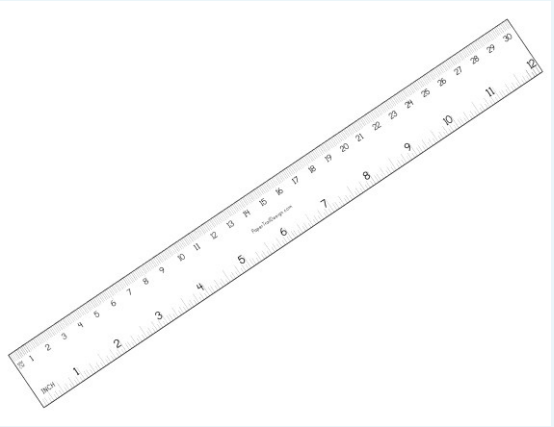
Read `ip26_l10_sample.txt`, then print:

- how many characters it contains
- whether it contains `"Café"`

Getting a `FileNotFoundError`? Run the notebook cell that writes the file first.

About 2 minutes · on your own, in the notebook

Example solution #2



```
file = open(sample_path, "r", encoding="utf-8")
contents = file.read()
file.close()

print(len(contents))
print("Café" in contents)
```

Output

```
35
True
```

Newlines count as characters too: three of the 35 are `\n`.

Predict: forgot to close?

We read the file back before and after `close()`. What do the two `print()` calls show?

```
draft_path = "ip26_l10_draft.txt"
file = open(draft_path, "w", encoding="utf-8")
file.write("Remember to close me\n")

reader = open(draft_path, "r", encoding="utf-8")
print(repr(reader.read()))
reader.close()
file.close()

reader = open(draft_path, "r", encoding="utf-8")
print(repr(reader.read()))
reader.close()
```

Output

```
''
'Remember to close me\n'
```

Until a file is closed, what we wrote may still be in memory.

A crash before `close()` can lose it.

! An error skips close(): use `with`

```
file = open("ip26_l10_numbers.txt", "w", encoding="utf-8")
file.write(42)
file.close()
print(file.closed)
```

→ `TypeError: write() argument must be str, not int`
→ `False` ← the error stopped the cell: `file.close()` never ran

```
with open("ip26_l10_numbers.txt", "w", encoding="utf-8") as file:
    file.write(42)
print(file.closed)
```

→ `TypeError: write() argument must be str, not int`
→ `True` ← `with` closes the file, **even after an error**

Reading one line at a time

A file object is *iterable*: each turn of the loop gets **one line**, as a string.

```
with open(sample_path, "r", encoding="utf-8") as file:
    for line in file:
        print(repr(line))
```

Output

```
'First line\n'
'Second line\n'
'Café: £2.50\n'
```

A line you read is a *string, so y*ou can postprocess it (strip, split, …) to save only the information you want.

```
raw = "   Portobello Beach   \n"
print(repr(raw))
clean = raw.strip()
print(repr(clean))
```

Output

```
'   Portobello Beach   '
'Portobello Beach'
```

`.strip()` removes spaces, tabs, and newlines from both ends

Try it yourself #3: count non-empty lines

We write a file with three lines of text and one **blank** line.

```
lines_path = "ip26_l10_lines.txt"
with open(lines_path, "w", encoding="utf-8") as file:
    file.write("alpha\n\nbeta\ngamma\n")
```

Read it back, one line at a time, and count only the lines that are not empty.

About 3 minutes

Example solution #3

```
lines_path = "ip26_l10_lines.txt"
with open(lines_path, "w", encoding="utf-8") as file:
    file.write("alpha\n\nbeta\ngamma\n")

count = 0
with open(lines_path, "r", encoding="utf-8") as file:
    for line in file:
        if len(line.strip()) > 0:
            count += 1
print(count)
```

→ 3

Writing and appending

Mode	File exists	File missing
`"r"`	read its contents	`FileNotFoundError`
`"w"`	**erase** it, then write	create it
`"a"`	add at the end	create it

Watch out: `"w"` empties an existing file the moment it is opened, even if you never write to it.

```
log_path = "ip26_l10_study_log.txt"
with open(log_path, "w", encoding="utf-8") as file:
    file.write("Started loops practice\n")
    file.write("Completed three exercises\n")

with open(log_path, "a", encoding="utf-8") as file:
    file.write("Reviewed collections\n")

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

Predict: how many entries are printed? And if you run the cell twice?

Output

```
Started loops practice
Completed three exercises
Reviewed collections
```

Text in, text out

```
readings_path = "ip26_l10_readings.txt"
total = 0
with open(readings_path, "r", encoding="utf-8") as file:
    for line in file:
        total += line
```

→ `TypeError: unsupported operand type(s) for +=: 'int' and 'str'`

Convert with `int()` or `float()` first!

```
total = 0
count = 0
with open(readings_path, "r", encoding="utf-8") as file:
    for line in file:
        value = int(line.strip())
        total += value
print(f"total = {total}")
```

→ `total = 36`

`ip26_l10_readings.txt`

```
12
15
9
```

Watch out: a file of numbers is still a file of text!

{ Live coding: a high-score file

Code along with me in a new cell:

- Save two scores to a file, then append a third
- **Predict:** what does `max(["120", "95", "140"])` return?
- Read the scores back and find the winner

`ip26_l10_scores.txt`

```
Maya 120
Sam 95
Ali 140
```

> Summary

- `\n` and `\t` are single characters, and `repr()` shows them
- Absolute paths start at the root, relative paths at the working directory
- Windows paths use `\`: in Python strings, write them with `/`
- `with open(...) as file:` closes the file for us, even after an error
- `"r"` read, `"w"` replace, `"a"` add at the end, always with `encoding="utf-8"`
- `file.read()` gives one string, `for line in file:` one line at a time
- Lines keep their `\n`: `.strip()` it
- Text from files is text: convert with `int()` or `float()`

Homework: the consolidation challenge at the end of the notebook.