# INFO5002: Intro to Python for Info Sys

## I/O

Slides created by: Zachary Doucet


Northeastern University

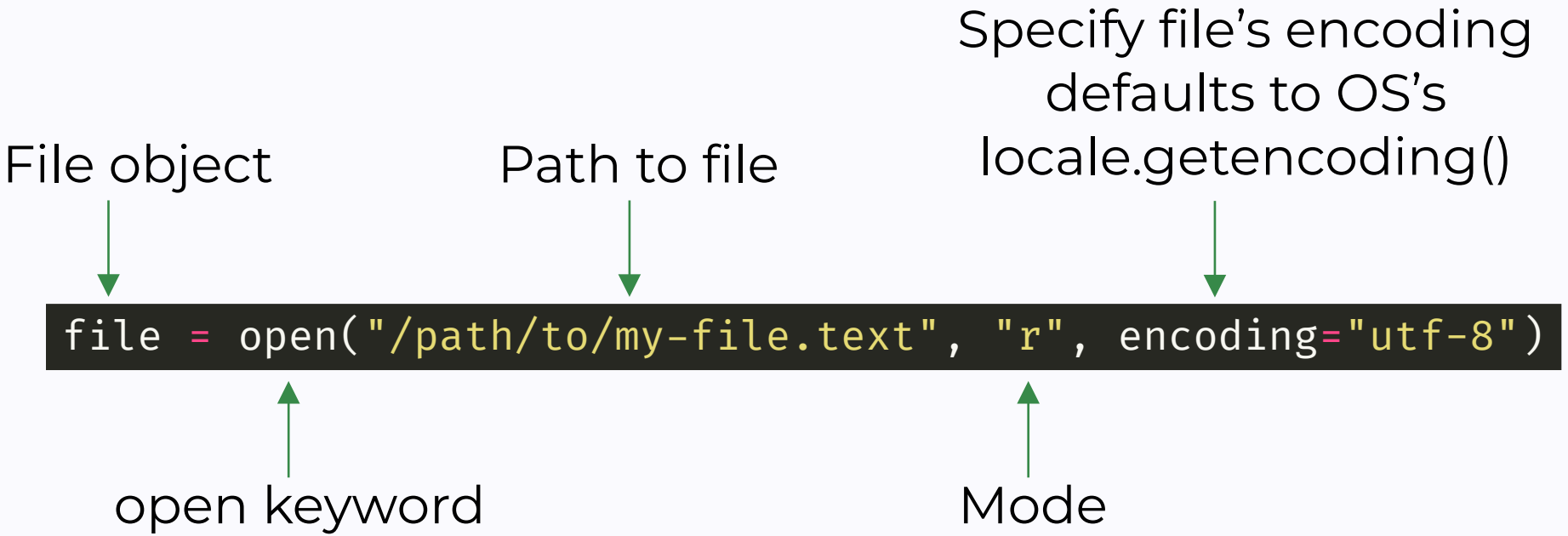
# So far all data has been in memory

- We have been creating variables that hold data in our code.
- This data is not persistent and when the program stops running all of the data is released.

# Keep persistence with IO

- Input/Output (I/O) acts as an **interface** between your code and the operating system's file system.
- Three types:
  - text (string)
  - binary (bytes): can store non-text data
  - Raw: rarely used

# Load text with open r



# File modes

| Character | Meaning |
|---|---|
| r | reading (default) |
| w | writing, deleting old first |
| x | create new file, fail if already existing |
| a | writing, appending to old |
| b | binary mode |
| t | text mode (default) |
| + | suffix, to add on read or write<br>r+ (read and write w/out deleting file first)<br>w+ (read and write deleting file first) |

# Encodings

- Important to specify the file's encoding that you are loading. Otherwise your data will be unrecognisable at worst or crash at best.
- Popular encodings: ascii, utf-8, utf-16, utf-32.

# Encoding matters example

```
file = open("test_file.txt", "w", encoding="utf-8")
file.write("Welcome to class")
file.close()

file2 = open("test_file.txt", "r", encoding="utf-8")
print(file2.read())
file2.close()

file3 = open("test_file.txt", "r", encoding="utf-16")
print(file3.read())
file3.close()
```

```
  File "/usr/lib64/python3.13/encodings/utf_16.py", line 67, in _buffer_decode
    raise UnicodeDecodeError("utf-16", input, 0, 2, "Stream does not start with BOM")
UnicodeDecodeError: 'utf-16' codec can't decode bytes in position 0-1: Stream does not start with BOM
```

# Load binary with open rb

```
file = open("/path/to/my-file.text", "rb")
```

# Load raw with open rb no buffering

```
file = open("/path/to/my-file.text", "rb", buffering=0)
```

# What to do with file objects?

- close(): close the file
- readline(): read one line from file
- readlines(): return list of lines from file
  - for line in file:
- read(size=-1): read size number of bytes
- write(data): write data

# Move file cursor

- seek(offset, whence=os.SEEK_SET): change the stream number of bytes from relative position set with whence.
  - os.SEEK_SET: start of stream (offset >= 0)
  - os.SEEK_CUR: current position
  - os.SEEK_END: end of stream (offset <= 0, usually)

# Let’s practice

Create a function **process_student_grades** that loads a file you will find on canvas and creates a new file **output.txt** which has number of 0-9% on one line, number of 10-19% on one line, ..., number of 90-99% on one line, number of 100% on one line, and the average on the final line. Use utf-8 encoding.

# It is annoying have one data per line!

- There are two commonly used ways to store data

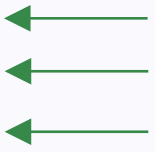
  1. CSV (Comma Separated Values)

  2. JSON (Javascript Object Notation)

# CSV

- Good for simple data.
- Created by separating values with a comma.

```
id,name,num_runways,num_gates
YVR,"Vancouver International Airport",3,101
YXX,Abbotsford International Airport,2,
```



Optional header
Full entry
Entry missing num_gates

# Reading CSV files

- Import csv module and create a csv reader with a file object.

```
import csv

file = open("csv-1.csv", "r")
reader = csv.reader(file)
header = next(reader)
for line in reader:
    print(line)
```

← Create csv reader
← Header first line (if exists)
← For all the other lines
← Do something here

- But for the **line** you *need to know* which index the data is.

# Use DictReader

- Same as regular reader but will have each line be a dictionary instead of list with the keys as the header (or anything else you specify).

```
import csv

file = open("csv-1.csv", "r")
reader = csv.DictReader(file)
for line in reader:
    print(line["num_runways"])
```

Create csv dictionary reader

For all lines

Do something here

# Similarly can write

```
import csv

file = open("csv-1.csv", "w")
writer = csv.writer(file)
writer.writerow(["id", "name", "num_runways", "num_gates"])
writer.writerow(["YVR", "Vancouver Intl Airport", 3, 101])
```

```
import csv

file = open("csv-1.csv", "w")
fieldnames = ["id", "name", "num_runways", "num_gates"]
writer = csv.DictWriter(file, fieldnames=fieldnames)
writer.writeheader()
writer.writerow({"id": "YVR", "name": "Vancouver Intl
Airport", "num_runways": 3, "num_gates": 101})
```

# Let’s practice

Create a class **Weather** which has a constructor that takes a path to the daily temperature CSV file and loads the data. Create a few functions:

- **average_high**: which returns the average high (deg C).
- **average_low**: which returns the average low (deg C).
- **average_max_gust**: which returns avg max gust (km/hr).

All functions optionally take **month** to specify for a month.

# JSON

- Good for complex structured data.
- Javascript Object Notation which features a Python dictionary-like structure.

```
{
    "airports": [
        {
            "id": "YVR",
            "name": "Vancouver Intl Airport",
            "num_runways": 3,
            "num_gates”: 101,
        }
    ]
}
```

# Write with json dump

```
import json

airports = [
    {"id": "YVR", "name": "Vancouver Intl
Airport", "num_runways": 3, "num_gates": 101}
]

file = open("data.json", "w")
contents = json.dump({"airport": airports},
file)
file.close()
```

# Read with json load

```
import json

file = open("data.json", "r")
contents = json.load(file)
print(contents)
file.close()
```

```
{'airport': [{'id': 'YVR', 'name': 'Vancouver Intl Airport',
'num_runways': 3, 'num_gates': 101}]}
```

# Let’s practice

Create a class **Client** which has a constructor that will ask for the user’s first name, last name, date of birth, and phone number if it has not already asked and the data file does not exist. If exists load the data to the right attributes.