COMP10001

WORKSHOP #7

Shevon Mendis
shevonm@unimelb.edu.au

Files

- Allow us to store data on a computer
    - The data stored is permanent: it will persist on the storage device after the program has terminated

Working with Files

- Opening a file:

```
fp = open(path_to_file, file_mode)
```

  - `fp` denotes a *file pointer*, which is essentially a reference to the file in the memory
- File modes:
  - `'r'`
    - Denotes that file will only be read (from the start)
    - Default value for the file_mode argument
  - `'w'`
    - Denotes that the file will only be written to
    - If the file already has data, that data will be deleted
  - `'a'`
    - Denotes that the file will be appended to (ie. data will be added to an already an already existing file)

Working with Files

- Useful file methods:
  - `.read()`
    - Reads the entire file and returns its contents as a string
  - `.readline()`
    - Reads a single line from the file and returns it as a string
  - `.readlines()`
    - Reads the entire file and returns a list where each element corresponds to a line in the file
  - `.write(line: str)`
    - Takes a `str` argument and writes that to the end of the file
      - **Note:** Does **not** add a `'\n'` character at the end of the line being written
  - `.writelines(lines: list)`
    - Takes a `list` of `str`s as an argument and iteratively writes them to the end of the file

Working with Files

- Closing a file:

  `fp.close()` → Flushes any unwritten information and closes the file object

- You should **always** close files when you're done with them
  - Not closing files can can slow down your program- too many files opens equates to more RAM being used, thereby hindering performance
  - For the most part, many changes to files in Python do not go into effect until *after* the file is closed, so if your script edits, leaves open, and reads a file, it won't see the edits. the data is often cached in memory and isn't written to the storage media until the file is closed. The longer you keep the file open, the greater the chance that you will lose data
  - Some operating systems (Windows, in particular) treat open files as locked and private- while you have a file open, no other program can also open it, even just to read the data #MacOSMasterRace

CSV Files

- A CSV (Comma Separated Values) file is a text file where the information is organised in a format similar to that of a spreadsheet

band_members.csv

```
Name,Position,Nickname
Dewey Finn,Lead Singer/ Guitarist,Mr.S
Summer Hathaway,Band Manager,Tinkerbell
Zack Mooneyham,Lead Guitarist,Zack-Attack
Freddy Jones,Drummer,Spazzy McGee
Lawrence,Keyboardist,Mr.Cool
Billy,Band Stylist,Fancy Pants
```

- Each line (row) represents a record
  - The first row is *often* going to be a header, which has the names for the different fields
- Each column represents a record's value for a particular field
- Useful for storing information like statistics or measurement data

Working with CSVs

- `csv.reader(`*file*`)`
  - Takes an **open** CSV file as its argument, and returns a `_csv.reader` object
    - `_csv.reader` is an (iterable) object, similar to what `.readlines()` produces, but it also takes the additional step of splitting each line (record) into a list of strings
    - Has the ability to determine whether a comma is a delimiter or part of the text within a field

Working with CSVs

- Usage Example: `csv.reader()`

```
import csv

band_members = open("band_members.csv")
data = csv.reader(band_members)

header = next(data)
print(f"Fields: { header }")

for idx, row in enumerate(data):
    print(f"Row #{ idx + 1 }: { row }")

band_members.close()
```

```
Fields: ['Name', 'Position', 'Nickname']
Row #1: ['Dewey Finn', 'Lead Singer/ Guitarist', 'Mr.S']
Row #2: ['Summer Hathaway', 'Band Manager', 'Tinkerbell']
Row #3: ['Zack Mooneyham', 'Lead Guitarist', 'Zack-Attack']
Row #4: ['Freddy Jones', 'Drummer', 'Spazzy McGee']
Row #5: ['Lawrence', 'Keyboardist', 'Mr.Cool']
Row #6: ['Billy', 'Band Stylist', 'Fancy Pants']
```

Working with CSVs

- `csv.DictReader(`*file*`)`
  - Takes an **open** CSV file as its argument, and returns a `csv.DictReader` object
    - `csv.DictReader` contains each row as a dictionary rather than as a list

Working with CSVs

- Usage Example: `csv.DictReader()`

```
import csv

band_members = open("band_members.csv")
data = csv.DictReader(band_members)

fields = data.fieldnames
print(f"Fields: { fields }")

for idx, row in enumerate(data):
    print(f"Row #{ idx + 1 }:")
    for field in fields:
        print(f"\t{ field }: { row[field] }")

band_members.close()
```

```
Fields: ['Name', 'Position', 'Nickname']
Row #1
      Name: Dewey Finn
      Position: Lead Singer/ Guitarist
      Nickname: Mr.S
Row #2
      Name: Summer Hathaway
      Position: Band Manager
      Nickname: Tinkerbell
```

Working with CSVs

- `csv.writer(`*file*`)`
  - Takes an file **that was opened in write mode** as its argument, and returns a `_csv.writer` object
  - Useful methods for `_csv.writer`:
    - `.writerows(2d_data: list)`
      - Takes a list of lists (each representing a row) as an argument and writes each of those nested lists as a new row to the required file
    - `.writerow(row: list)`
      - Takes a list (representing a record) as an argument and writes it as a new row to the required file

Project 2 Reveal