Warmup (L15)

Create a function to sum *every* column in a CSV file, returning a dictionary associating each column name with the sum. You may assume that every column contains only numbers.

```
def sumEveryColumn(filename: str) -> dict[str, float]:
```

Writing files

CS114 M6

Writing

- So far we've been consuming data produced by others
- We've output data to the screen (`print`) and into variables, but not yet into files
- Let's change that!

Writing functions

- FileLib has the reverse of most functions we've seen so far:
  - `ls_to_txt`
  - `ld_to_csv`
  - `dl_to_csv`
  - `ll_to_csv`
- These take the filename and the data to write, and write in the same format as their reverses read
- WARNING: Python will happily overwrite files you already have! Make sure you know what you're writing to!

Reverse every line

Let’s write a function to copy text from one file to another, but reverse every line

Reverse every line

```
def reverseLines(
    outputFile: str, inputFile: str
) -> None:
    lines = txt_to_ls(inputFile)
    for idx in range(len(lines)):
        lines[idx] =
            lines[idx].strip()[::-1]
    ls_to_txt(outputFile, lines)
```

Reverse every line

- By default, `ls_to_txt` adds the line breaks itself
- Remember, `txt_to_ls` kept the line breaks, so if you're not careful, you'll double up on line breaks!
- .strip() saved us here, but let's look at what would've happened if we ask `ls_to_txt` *not* to add line breaks

Reverse every line

```
def reverseLines(
    outputFile: str, inputFile: str
) -> None:
    lines = txt_to_ls(inputFile)
    for idx in range(len(lines)):
        lines[idx] =
            lines[idx].strip()[::-1] + "\n"
    ls_to_txt(
        outputFile, lines,
        add_newline=False
    )
```

What's this thing???

Line-break Hell!

- This weird `\n` is an *escape sequence*: it means “interpret this as a line break, even though I’ve literally written a backslash and then an n”
  - (n stands for “new line”)
- Remember, `print` adds its own line breaks, and `ls_to_txt` adds its own line breaks, but if you need to make your own, `\n` is the way

Making better text

CS114 M6

Print is too smart

- `print` is very powerful
  - It can print any kind of data, adds its own newlines, etc.
- To write to a file, we need a bunch of strings
- `str(...)` will convert many things into a string, but it's limited
- Python has a better way of making strings

Formatted strings

```
def lineInfo(
    outputFile: str, inputFile: str
) -> None:
    lines = txt_to_ls(inputFile)
    for idx in range(len(lines)):
        lines[idx] =
f"Line length: {len(line)}. Line reversed:
{line.strip()[::-1]}\n"
    ls_to_txt(
        outputFile, lines,
    add_newline=False
)
```

New syntax!

- Format strings are written like strings (in quotes), but with an '**`f`**' before the opening quote
- In format strings, everything in braces (`{ }`) is run as Python code, and its result put in the string
- Extremely useful for writing to files, but useful anywhere you want something string'd

In-lecture quiz (L15)

- https://student.cs.uwaterloo.ca/~cs114/quiz/
- Q1: I tried to fix it but it's still printing double-spaced! Why?

```
f = txt_to_ls("file.txt")
for line in f:
    line.strip()
    print(line)
```

A. `strip` is the wrong method
B. A **`for`** loop isn't the right way to get lines
C. `line.strip()` doesn't change `line`
D. Need `f.strip()`, not `line.strip()`
E. Python is cruel and vindictive

In-lecture quiz (L15)

- https://student.cs.uwaterloo.ca/~cs114/quiz/
- Q2: What is the type of `x`?

```
x = csv_to_ld("nino34.csv")
```

A. This code has an error (no type)
B. `list[str]`
C. `list[dict[str, float]]`
D. `list[dict[str, str]]`
E. `dict[str, list[float]]`

Writing CSV files

CS114 M6

Writing CSV files

- Just like FileLib has writing functions for text files, it has writing functions for CSV files too
- To demonstrate, let's add a Fahrenheit field from nino34.csv into nino34f.csv

Writing CSVs

```
nino = csv_to_ld("nino34.csv")
for row in nino:
    row["Fahrenheit"] =
        str(float(row["TOTAL"])*9/5+32)
ld_to_csv("nino34f.csv", nino)
```

Other file formats

CS114 M6

JSON

- CSV is structured, but two dimensional
- What if you have complex data with lists and dictionaries and all sorts of stuff in it?
- JSON is a standard format for complicated data

Example JSON

```
{
    "surname": "Richards",
    "given name": "Gregor",
    "height": 1.76,
    "places lived": ["USA", "Canada"]
}
```

- What an unfamiliar way of storing data! Never seen anything like that!

JSON is from programming

- JSON looks like how you would store data in a programming language, because… that's what it is
- It's *JavaScript Object Notation*, and JavaScript is a different programming language
- Python's syntax for data is nearly the same
  - This is partially because they share heritage, and partially Python being influenced by JSON

What can JSON do

- Strings
- Numbers
- Booleans
- Lists of anything it can store (including lists)
- Dictionaries
  - Key must be a string
  - Values can be anything JSON can store

Using JSON

- FileLib has:
  - `json_to_any` (to load a JSON file)
  - `any_to_json` (to store data into a JSON file)
- ipynb files are JSON, so let's open our own exercise notebook to show JSON in action!

SSV

- Occasionally you'll see quasi-CSV files where the data is separated by spaces instead of commas
- Luckily, FileLib has parameters to handle this:

```
nino = csv_to_ld("nino34.ssv", delimiter=" ")
```

XML

- Pray you never encounter XML
  - You probably won't, but it's used in, e.g., PowerPoint files
- There's an `xml` module. We shall discuss it no further.

Binary formats

- There *are* file formats that aren't text
- Mostly used when the amount of data is huge
  - A 5MP picture is worth about 2.5M words
- If you find yourself needing to use one, search for the module that handles it; there'll usually be one!

Fun with files

CS114 M6

Word distribution CSV

- Let’s do another long demonstration
- We’ll continue our word distribution example from A Tale of Two Cities, but
  - fix our issues with non-word stuff,
  - sort the result by frequency, instead of by word, and
  - store the result in a CSV file.

Step one: find the words

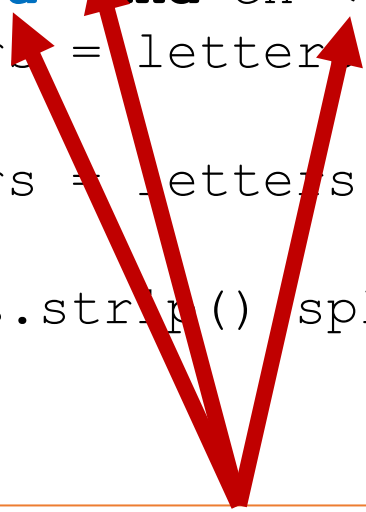
- We want a function that gets only the words from a string, and drops all the non-word symbols
- We actually have everything we need, but it may not be obvious how
- Biggest trick here: string comparison

Step one: find the words

```
def words(line: str) -> list[str]:
    letters = ""
    for ch in line.lower():
        if ch >= "a" and ch <= "z":
            letters = letters + ch
        else:
            letters = letters + " " # So we'll
                                    # still split
    return letters.strip().split()
```

Step one: find the words

```
def words(line: str) -> list[str]:
    letters = ""
    for ch in line.lower():
        if ch >= "a" and ch <= "z":
            letters = letters + ch
        else:
            letters = letters + " " # So we'll
                                    # still split
    return letters.strip().split()
```



Because we lower-cased first, we didn't need to check the capitals case.

Aside: remember `distribution`

```
def distribution(
    lst: typing.Sequence
) -> dict[typing.Any, int]:
    r = {}
    for val in lst:
        if not (val in r):
            r[val] = 0
        r[val] = r[val] + 1
    return r
```

Step two: sort by frequency

- We did this in a warmup!

```
def dictValue(key: str) -> int:
    return dist[key]

byFreq = sorted(dist, reverse=True, key=dictValue)
```

Step three: write to CSV

- Let’s put it together, into one function that takes the output (CSV) and input (text) file as arguments

```
def wordDistributionCSV(
    outFilename: str, inFilename: str
) -> None:
    # 1: Words
    inWords = words(txt_to_s(inFilename))

    # 2: Distribution and sorting
    dist = distribution(inWords)
    def dictValue(key: str) -> int:
        return dist[key]
    byFreq = sorted(dist, reverse=True, key=dictValue)

    # 3: Write CSV
    with open(outFilename, "w") as o
    data: list[dict[str, str]] = []
    for word in byFreq:
        data.append({
            "Word": word,
            "# of appearances": dist[word]
        })
    ld_to_csv(outFilename, data)
```

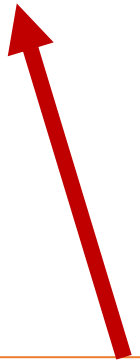
Another CSV demo

```
!wget https://student.cs.uwaterloo.ca/~cs114/src/employee_names.csv
!wget https://student.cs.uwaterloo.ca/~cs114/src/employee_wages.csv
```

- We have two CSV files:
  - Employee ID,Name
  - Employee ID,Hourly wage
- We want a single CSV file that has Employee ID,Name,Hourly Wage
- But, some data may be missing!

```
import typing

employeeInfo: dict[int, dict[str, typing.Any]] = {}

def mkEmployeeInfo(id: int) -> dict[str, typing.Any]:
    if not (id in employeeInfo):
        employeeInfo[id] = {"Employee ID": id}
    return employeeInfo[id]

for row in csv_to_ld("employee_names.csv"):
    id = int(row["Employee ID"])
    info = mkEmployeeInfo(id)
    info["Name"] = row["Name"]

for row in open("employee_wages.csv"):
    id = int(row["Employee ID"])
    info = mkEmployeeInfo(id)
    info["Hourly wage"] = row["Hourly wage"]
```

```
employeeCSV: list[dict[str, typing.Any]] = []
for employeeId in employeeInfo:
    info = employeeInfo[employeeId]
    if not ("Name" in info):
        info["Name"] = "!UNKNOWN!"
    if not ("Hourly wage" in info):
        info["Hourly wage"] = "!UNKNOWN!"
    employeeCSV.append(info)

ld_to_csv(employeeCSV)
```



This would be a pretty brutal way to add our unknowns if we had more columns than this. Can you think of another way?

Module summary

CS114 (M6)

Module summary

- Reading and writing text files
- Reading and writing CSV files using dictionaries and lists
- Reading and writing JSON data
- Moving data back and forth between files and Python values