

# LECTURE 20

## CSV AND JSON

MCS 275 Spring 2023

David Dumas

# LECTURE 20: CSV AND JSON

Reminders and announcements:

- Project 3 coming soon.

# INSTALL PILLOW

To prepare for Friday's lecture, you need the Python package Pillow. Most people can install it with

```
python3 -m pip install pillow
```

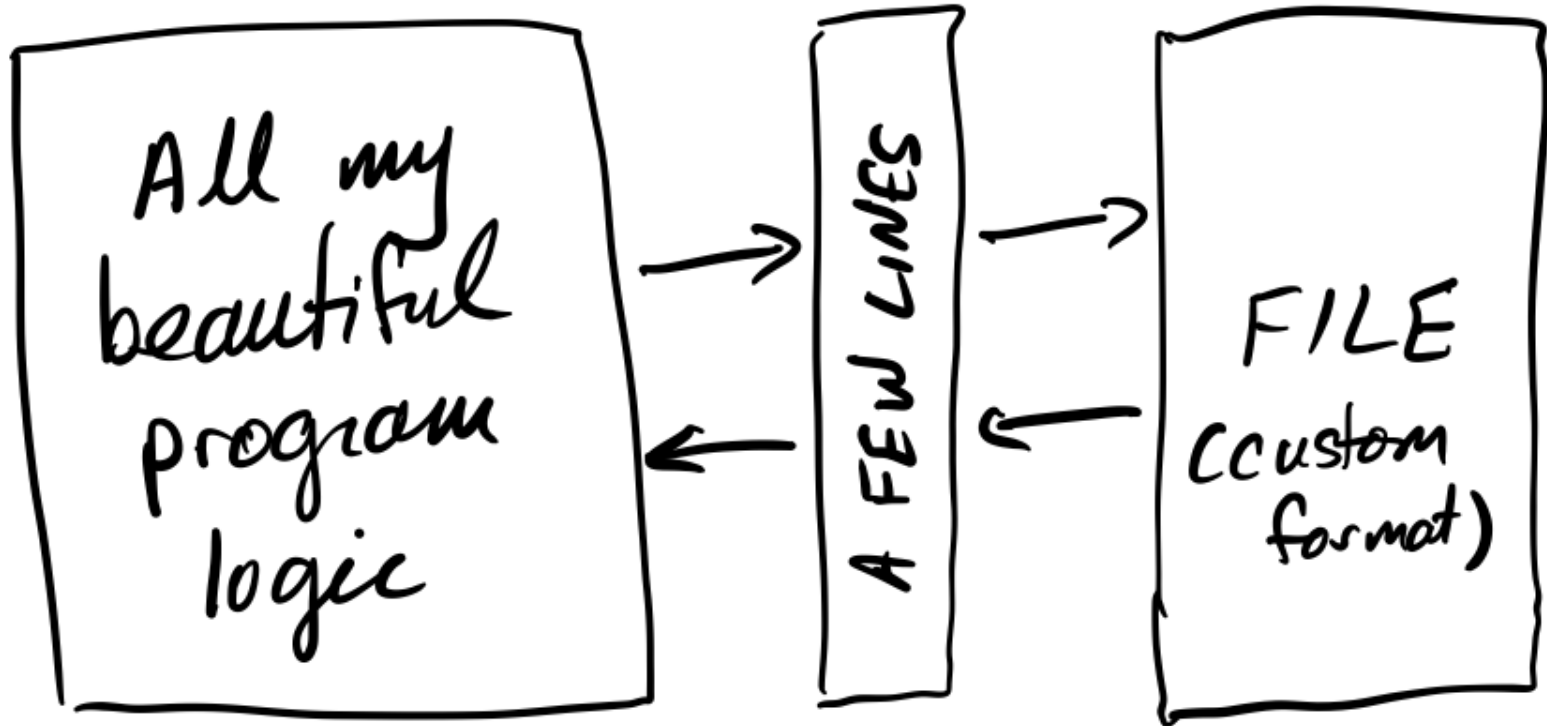
Or substitute the correct interpreter name for your platform.

If you have trouble, check the [install instructions](#) and let us know if you don't find a solution there.

# READING AND WRITING DATA FILES

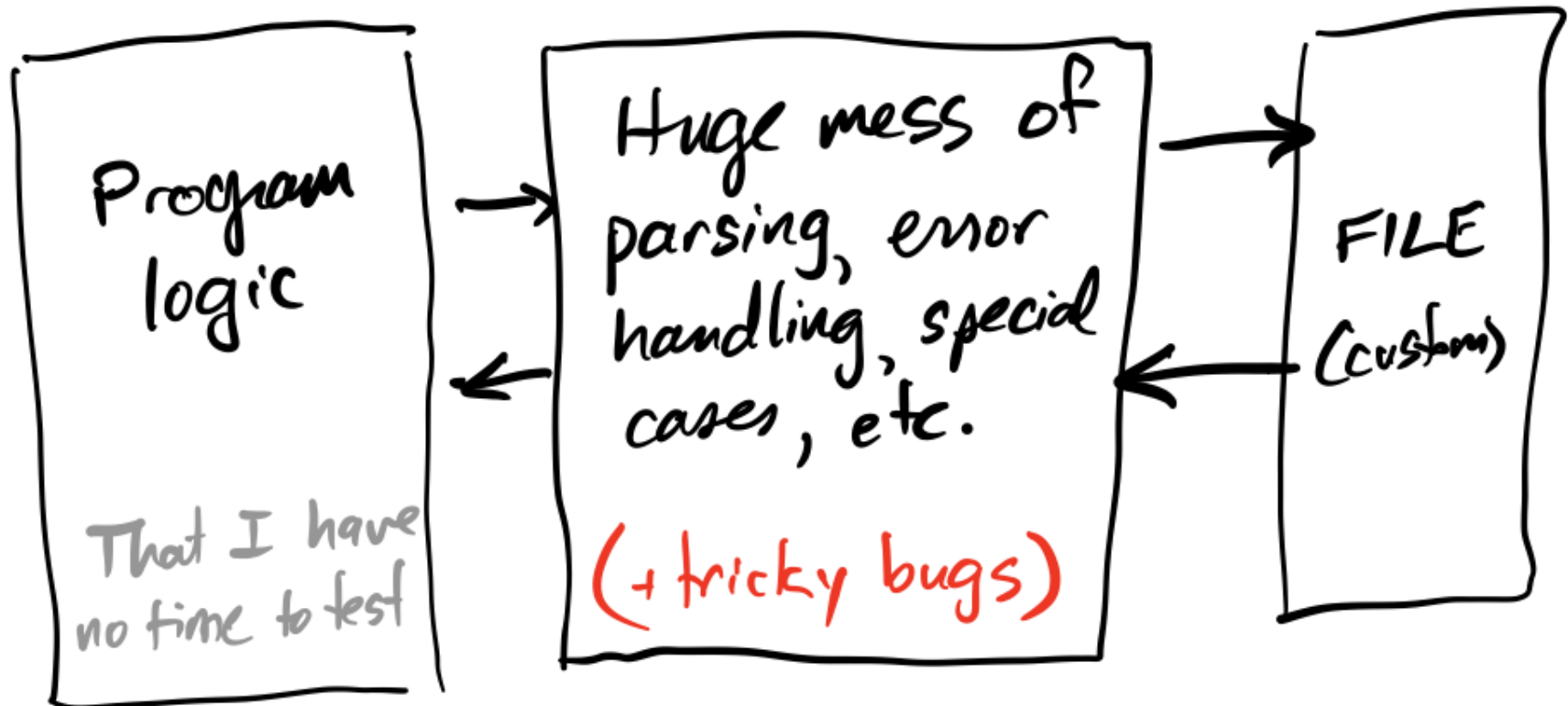
# READING AND WRITING DATA FILES

## THE VISION



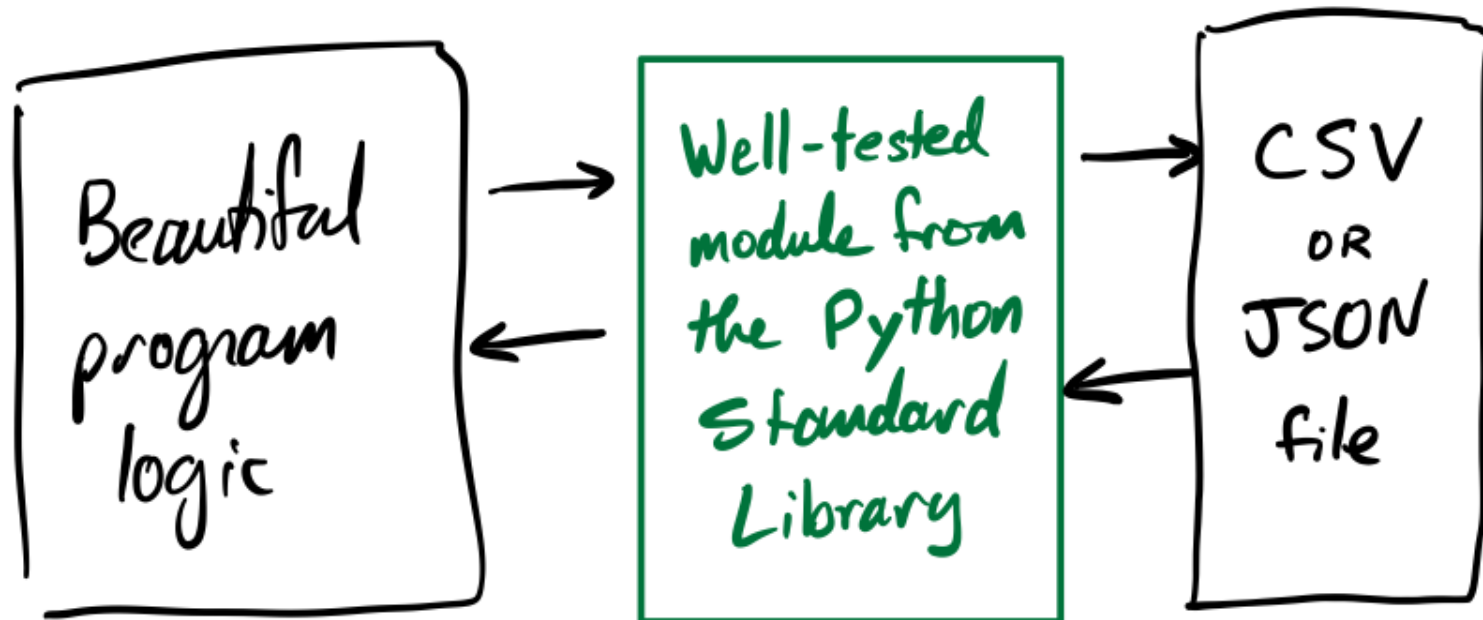
# READING AND WRITING DATA FILES

## THE REALITY



# READING AND WRITING DATA FILES

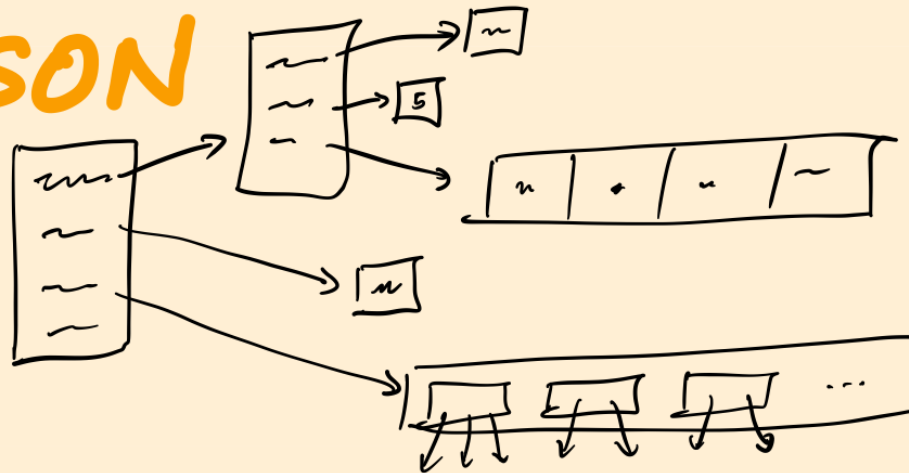
## THE SOLUTION



1	name	age	gender
2	1000-1234	25	male
3	name	age	gender
4	1000-1234	25	male
5	name	age	gender
6	1000-1234	25	male

CSV

JSON





# CSV

The 90% correct one-line summary:

*A way to store a spreadsheet in a text file.*

# CSV

Comma separated values. A text file format like:

```
State,Capital,Population  
Kentucky,Frankfort,25527  
South Dakota,Pierre,13646
```

Column headings in the first row (usually).

**Untyped.** Up to reader to figure out  
string/float/int/etc.

District	Fin-Sub	Chrgbl	Fin No	PO Name	Unit Name	Property Address	Cou
Greater Boston	431120	G01	431120	BARRINGTON	MAIN OFFICE	200 MIDDLE HW	
Greater Boston	432360	G01	432360	COVENTRY	MAIN OFFICE	1550 NOOSENECK	
Greater Boston	434480	G01	434480	HARRISVILLE	MAIN OFFICE	131 HARRISVI	
Greater Boston	436020	G01	436020	NEWPORT	MAIN OFFICE	320 THAMES ST ST	
Greater Boston	436090	G02	436090	NORTH KINGSTOWN	MAIN OFFICE	7715 POS	
Greater Boston	436580	G02	436580	PASCOAG	MAIN OFFICE	35 BRIDGE WAY, PR	
Greater Boston	436723	G01	436723	PAWTUCKET	CUMBERLAND BR.	2055 DIAMON	
Greater Boston	436720	G03	436720	PAWTUCKET	DARLINGTON	30 MONTICELLO R	
Greater Boston	436720	G01	436720	PAWTUCKET	MAIN OFFICE	40 MONTGOMERY	
Greater Boston	436720	G01	436720	PAWTUCKET	MAIN OFFICE	40 MONTGOMERY	
Greater Boston	436860	G01	436860	PORTSMOUTH	MAIN OFFICE	95 CHASE RD, N	
Greater Boston	437140	G07	437140	PROVIDENCE	CORLISS PK. STA & VMF	55	
Greater Boston	437140	G07	437140	PROVIDENCE	CORLISS PK. STA & VMF	55	
Greater Boston	437178	G01	437178	PROVIDENCE	EAST PROVIDENCE BR.	17 GR	
Greater Boston	437166	G01	437166	PROVIDENCE	JOHNSTON BRANCH	1530 ATWO	
Greater Boston	437170	G01	437170	PROVIDENCE	OLNEYVILLE STA	100 HARTFO	
Greater Boston	437141	G08	437141	PROVIDENCE	P&DC	24 CORLISS ST RM 100	
Greater Boston	437141	G08	437141	PROVIDENCE	P&DC	24 CORLISS ST RM 100	
Greater Boston	437141	G08	437141	PROVIDENCE	P&DC	24 CORLISS ST RM 100	
Greater Boston	438260	G07	438260	WAKEFIELD	MAIN OFFICE	551 KINGSTOWN	
Greater Boston	438260	G01	438260	WAKEFIELD	NARRAGANSETT BR.	15 MEMORI	

# READING CSV

```
with open("datafile.csv", "r", newline="", encoding="UTF-8") as f:
    rdr = csv.DictReader(f)
    rownum = 1
    for row in rdr: # reader objects are iterable (ONCE!)
        # row is a dict like {"State": "Kentucky", ...}
        print("Row", rownum)
        rownum += 1
        for colname in row:
            print("{}: {}".format(colname, row[colname]))
```

# WRITING CSV

```
with open("courses.csv", "w", newline="", encoding="UTF-8") as fp:
    w = csv.DictWriter(fp, fieldnames=["course", "instructor"])
    # Write the column headers
    w.writeheader()
    # Now write the rows of data
    w.writerow({"course": "MCS 260",
                "instructor": "Dumas"})
    w.writerow({"course": "MCS 275",
                "instructor": "Dumas"})
```

# WRITING CSV

```
with open("courses.csv", "w", newline="", encoding="UTF-8") as fp:
    w = csv.writer(fp)
    # Write the column headers
    w.writerow(["course", "instructor"])
    # Now write the rows of data
    w.writerow(["MCS 260", "Dumas"])
    w.writerow(["MCS 275", "Dumas"])
```

# JSON

The 90% correct one-line summary:

*A way to store a dict in a file.*

# JSON

JSON stands for **JavaScript object notation**. It is a text-based format for **typed** hierarchical data.



```
{  
  "title": "Fighting robotic wasps",  
  "authors": [  
    "Paolo Cortázar"  
  ],  
  "year": 2426,  
  "tags": [  
    "nonfiction",  
    "self-help"  
  ],  
  "credits": {  
    "editor": "Fayez Okoye-Sarkis",  
    "cover design": "Teresa Duarte"  
  },  
  "checked out": true,  
  "avg star rating": 4.89  
}
```

```
{
  "newsFeedItemList": [
    {
      "title": "Illinois Ranks #2 State in the Nation for Corporate
      "type": "Press Release",
      "date": "Wednesday, March 01",
      "year": "2023",
      "description": "CHICAGO – Site Selection Magazine – an intern
      "thumbnail": "https://www2.illinois.gov/IISNewsImages/rollupi
      "url": "https://www.illinois.gov/news/press-release.26121.htm
      "altText": ""
    },
    {
      "title": "State of Illinois Launches Coalition in Effort to E
      "type": "Press Release",
      "date": "Tuesday, February 28",
      "year": "2023",
      "description": "CHICAGO – The Illinois Department of Public H
      "thumbnail": "https://www2.illinois.gov/IISNewsImages/rollupi
      "url": "https://www.illinois.gov/news/press-release.26117.htm
      "altText": ""
    },
  ],
}
```

Source: [illinois.gov](https://www.illinois.gov) home page

# JSON VALUE TYPES

- **string** — must use double quotes.
- **number** — float, int, other? Up to reader.
- **boolean** — lower case names `true`, `false`.
- **null** — like Python `None`.
- **array** — like Python `list`. Brackets and commas.
- **object** — like Python `dict`. Curly braces, colons, and commas. Keys must be strings.

# READING JSON

```
with open("in.json", "r", encoding="UTF-8") as fp:  
    val = json.load(fp) # read from file  
  
# OR if you have a string  
val = json.loads(s)
```

The object returned can be hard to use if you don't have documentation for the layout of the file. But since it has keys and values, it is at least explorable.

# WRITING JSON

```
val = {  
    "temperature": 451.3,  
    "primes": [2,3,5,7,11],  
    "awesome": True,  
    "starter": "charmander"  
}  
with open("out.json", "w", encoding="UTF-8") as fp:  
    json.dump(val, fp) # save exactly one object to file  
  
# OR if you just want the JSON as a string  
s = json.dumps(val)
```

# KEY JSON FEATURES

- Does not require data to be tabular.
- Has excellent standardization and cross-language support.
- Most HTTP APIs (e.g. data portals) return JSON.
- Semi-readable and semi-writable for humans.

## Conversion table for Python → JSON

- `dict` → `object`
- `list` or `tuple` → `array`
- `int` or `float` → `number`
- `bool` → `boolean`
- `None` → `null`

# REFERENCES

- MCS 260 Fall 2021:
  - [Lecture 15: JSON](#)
  - [Lecture 16: CSV](#)
- [csv module documentation](#)
- [json module documentation](#)
- [Awesome JSON data sets](#)

# REVISION HISTORY

- 2022-03-04 Last year's lecture on this topic finalized
- 2023-03-01 Updated for 2023



