

# LECTURE 23

## CSV AND JSON

MCS 275 Spring 2022

David Dumas

# LECTURE 23: CSV AND JSON

## Course bulletins:

- Homework 8 posted.
- [Project 3](#) posted. Check it out. Brief discussion in Monday's lecture.
- Project 2 solutions posted. **Please read.**
- Project 2 will be graded by Monday.

# INSTALL PILLOW

Next week: Manipulating images with the Python package *Pillow*. To prepare, please

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

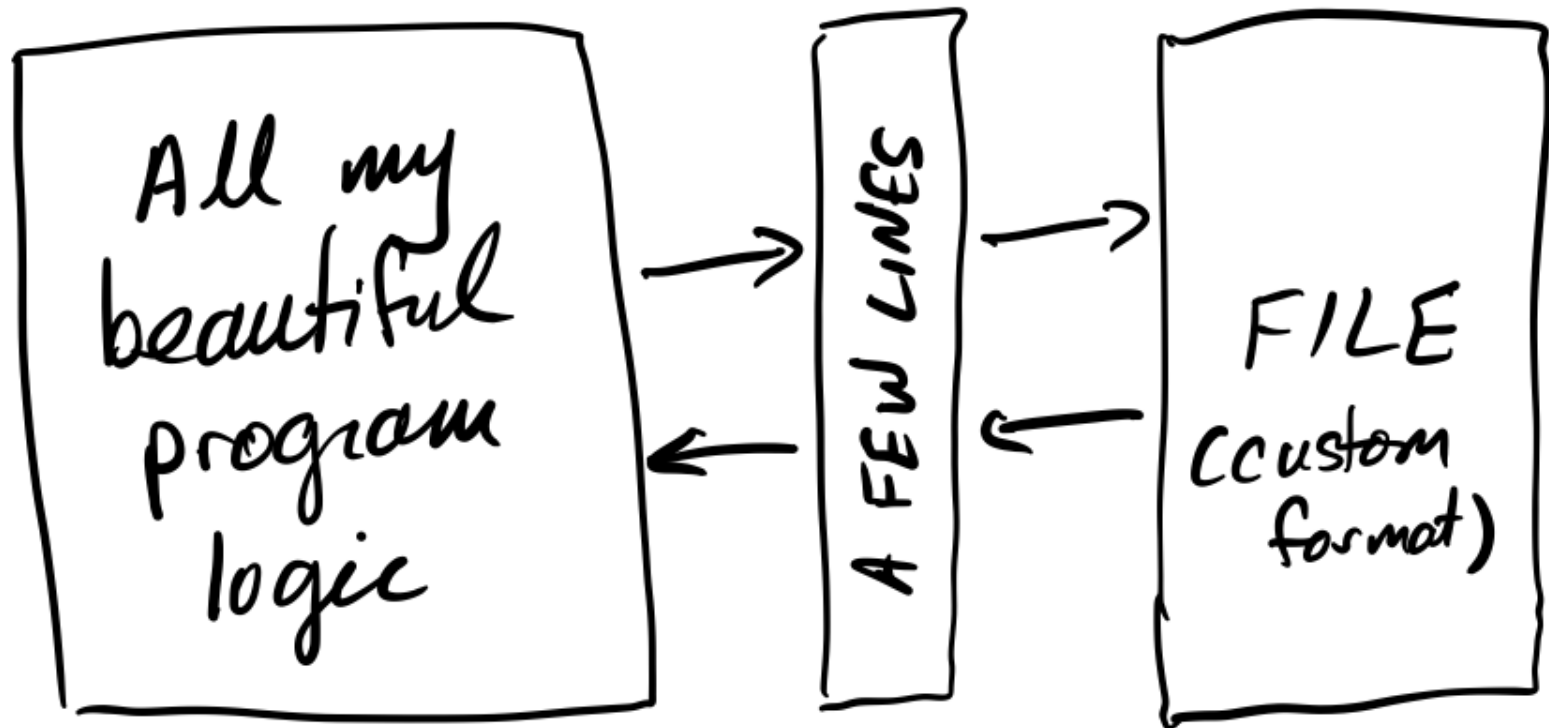
Or substitute the correct interpreter name for your platform.

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

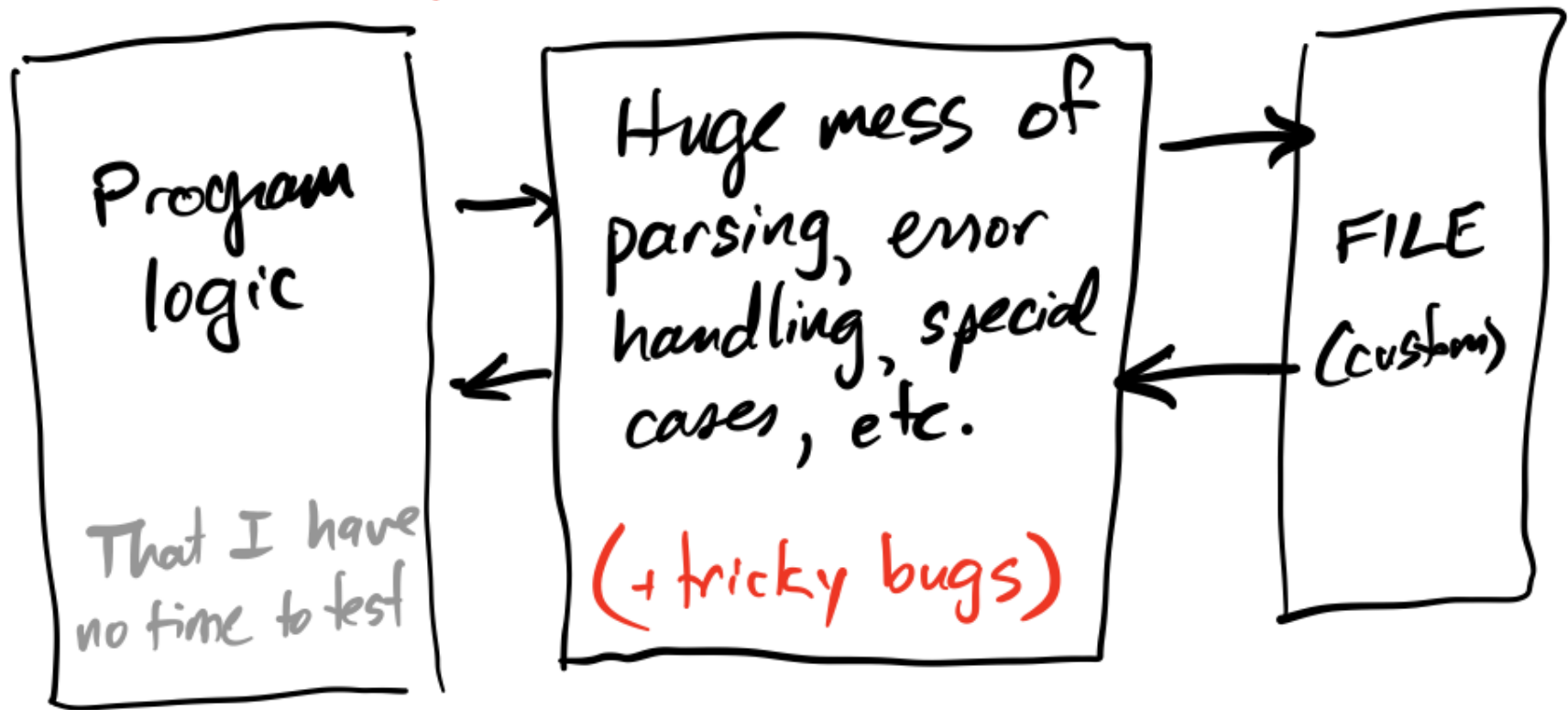
# READING AND WRITING DATA FILES

## THE VISION



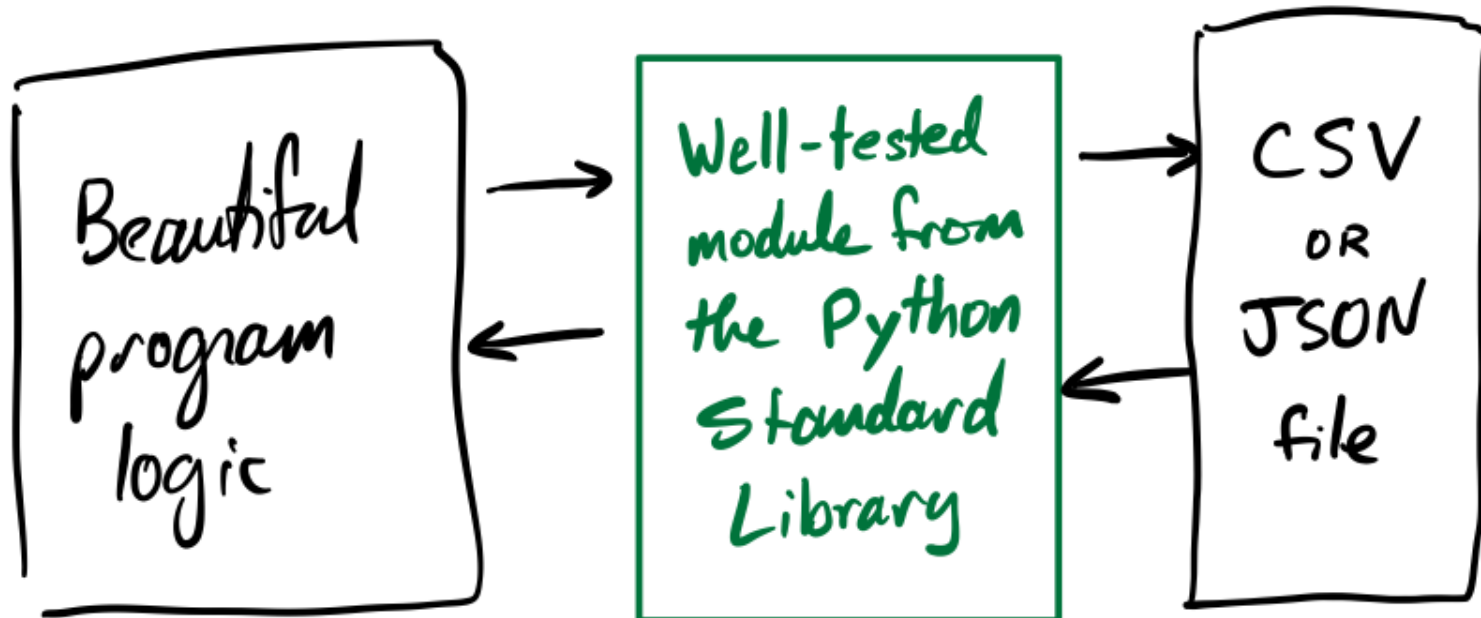
# READING AND WRITING DATA FILES

## THE REALITY



# READING AND WRITING DATA FILES

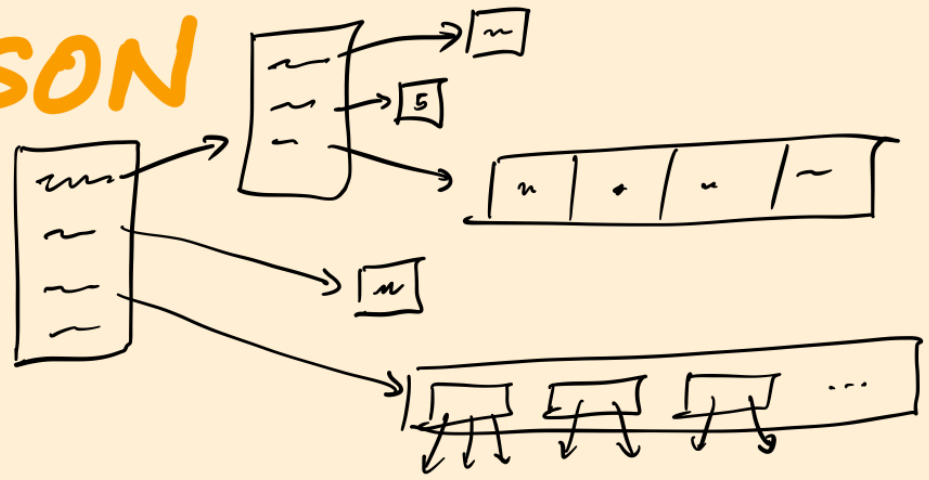
## THE SOLUTION



1	100	100	100
2	100	100	100
3	100	100	100
4	100	100	100
5	100	100	100
6	100	100	100

CSV

JSON





# 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	County
Greater Boston	431120-G01	431120	BARRINGTON	MAIN OFFICE	200 MIDDLE HWY	BI	
Greater Boston	432360-G01	432360	COVENTRY	MAIN OFFICE	1550 NOOSENECK HILL		
Greater Boston	434480-G01	434480	HARRISVILLE	MAIN OFFICE	131 HARRISVILLE		
Greater Boston	436020-G01	436020	NEWPORT	MAIN OFFICE	320 THAMES ST STE 1		
Greater Boston	436090-G02	436090	NORTH KINGSTOWN	MAIN OFFICE	7715 POST RD		
Greater Boston	436580-G02	436580	PASCOAG	MAIN OFFICE	35 BRIDGE WAY	PROVI	
Greater Boston	436723-G01	436723	PAWTUCKET	CUMBERLAND BR.	2055 DIAMOND H		
Greater Boston	436720-G03	436720	PAWTUCKET	DARLINGTON	30 MONTICELLO RD	PA	
Greater Boston	436720-G01	436720	PAWTUCKET	MAIN OFFICE	40 MONTGOMERY ST	PA	
Greater Boston	436720-G01	436720	PAWTUCKET	MAIN OFFICE	40 MONTGOMERY ST	PA	
Greater Boston	436860-G01	436860	PORTSMOUTH	MAIN OFFICE	95 CHASE RD	NEWPO	
Greater Boston	437140-G07	437140	PROVIDENCE	CORLISS PK. STA & VMF	55 CORLISS ST	PRO	
Greater Boston	437140-G07	437140	PROVIDENCE	CORLISS PK. STA & VMF	55 CORLISS ST	PRO	
Greater Boston	437178-G01	437178	PROVIDENCE	EAST PROVIDENCE BR.	17 GROVE ST	PRO	
Greater Boston	437166-G01	437166	PROVIDENCE	JOHNSTON BRANCH	1530 ATWOOD AVE	PRO	
Greater Boston	437170-G01	437170	PROVIDENCE	OLNEYVILLE STA	100 HARTFORD AVE	PRO	
Greater Boston	437141-G08	437141	PROVIDENCE	P&DC	24 CORLISS ST RM 100	PRO	
Greater Boston	437141-G08	437141	PROVIDENCE	P&DC	24 CORLISS ST RM 100	PRO	
Greater Boston	437141-G08	437141	PROVIDENCE	P&DC	24 CORLISS ST RM 100	PRO	
Greater Boston	438260-G07	438260	WAKEFIELD	MAIN OFFICE	551 KINGSTOWN RD	W	
Greater Boston	438260-G01	438260	WAKEFIELD	NARRAGANSETT BR.	15 MEMORIAL S	W	
Greater Boston	438540-G01	438540	WARREN	MAIN OFFICE	53 CHILD ST	BRISTOL, W	

# READING CSV

```
with open("datafile.csv", "r", newline="", encoding="UTF-8") as fp:
    rdr = csv.DictReader(fp)
    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

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

```
{  
  "title": "Fighting robotic wasps",  
  "author": "Paolo Cortázar",  
  "year": 2026,  
  "tags": ["nonfiction", "self-help"],  
  "checked out": true,  
  "avg star rating": 4.89  
}
```

```
{
  "newsFeedItemList": [
    {
      "title": "Planning and research grants available through IDOT ",
      "type": "Press Release",
      "date": "Thursday, March 03",
      "year": "2022",
      "description": "SPRINGFIELD - The Illinois Department of Transporta",
      "thumbnail": "https://www2.illinois.gov/IISNewsImages/rollupimage",
      "url": "/content/soi/illinois/en/news/press-release.24573.html",
      "altText": ""
    },
    {
      "title": "2021 Marion County Final Multiplier Announced",
      "type": "Press Release",
      "date": "Thursday, March 03",
      "year": "2022",
      "description": "SPRINGFIELD, IL, - Marion County has been issued",
      "thumbnail": "https://www2.illinois.gov/IISNewsImages/rollupimage",
      "url": "/content/soi/illinois/en/news/press-release.24572.html",
      "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-writeable 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

- 2021-03-04 Initial publication