

# Input/Output

Jamie Saxon

Introduction to Programming for Public Policy

October 11, 2016

# Introduction

- ▶ Our python scripts have been ‘self-contained.’
- ▶ We’d like to be able to use outside data.
- ▶ Today’s class will have a tiny bit of new material, and is mainly applications of dictionaries, lists, and for loops.
- ▶ In Week 5, we’ll learn the python module for “comfortably” using these types of data: Pandas.

## You have already run for loops over a file:

---

```
#!/usr/bin/env python
```

```
for line in open("salaries.csv", "r"):
    print(line.strip())
```

---

- ▶ The syntax is a for loop – nothin' to it!
- ▶ Just `open()` the file with a 'r' flag, for 'read.'
- ▶ You get one line at a time, and can do whatever you want with them.
- ▶ Use `strip()` to remove whitespace and `split()` to list-ify lines.

**Good news: very similar, but with “w”.**

```
#!/usr/bin/env python
```

```
yum = ["pineapple", "watermelon", "blueberry",  
       "apricot", "chirimoya", "grapefruit",]
```

```
output = open("output.txt", "w")  
for y in yum: output.write(y + "\n")  
output.close()
```

- ▶ The difference is that we're iterating over something else.
- ▶ The output file is just an object, that we write to.
- ▶ There is also 'a', for 'append' (write at end of file).

- ▶ You can also use 'with' to specify a block in which to write.
- ▶ The file 'snaps shut' at the end of the block.

---

```
#!/usr/bin/env python
```

```
yum = ["pineapple", "watermelon", "blueberry",  
       "apricot", "chirimoya", "grapefruit"]
```

```
with open("output.txt", "w") as output:  
    for y in yum: output.write(y + "\n")
```

---

# Formats

# CSV: Comma Separated Values

- ▶ Common, simple, flat, but non-standardized format.
  - ▶ Text in columns separated by a delimiter ('escape' by quotes).
  - ▶ Can be read directly by e.g., Excel.
  - ▶ There are python tools for this too (of course)
- ▶ We'll work with it by hand, to understand files and lists a bit better.

```
Name,Job Titles,Department,Full or Part-Time,Salary or Hourly,Typical Hours,Annual Salary,Hourly Rate
"AARON, JEFFERY M",SERGEANT,POLICE,F,Salary,,,$101442.00,
"AARON, KARINA ",POLICE OFFICER (ASSIGNED AS DETECTIVE),POLICE,F,Salary,,,$94122.00,
"AARON, KIMBERLEI R",CHIEF CONTRACT EXPEDITER,GENERAL SERVICES,F,Salary,,,$101592.00,
"ABAD JR, VICENTE M",CIVIL ENGINEER IV,WATER MGMNT,F,Salary,,,$110064.00,
"ABARCA, EMMANUEL ",CONCRETE LABORER,TRANSPORTN,F,Hourly,40,,,$36.18
"ABASCAL, REECE E",TRAFFIC CONTROL AIDE-HOURLY,OEMC,P,Hourly,20,,,$19.86
"ABBASI, CHRISTOPHER ",STAFF ASST TO THE ALDERMAN,CITY COUNCIL,F,Salary,,,$50436.00,
"ABBATACOLA, ROBERT J",ELECTRICAL MECHANIC,AVIATION,F,Hourly,40,,,$46.10
"ABBATE, JOSEPH L ",POOL MOTOR TRUCK DRIVER,STREETS & SAN,F,Hourly,40,,,$35.60
"ABBATEMARCO, JAMES J",FIRE ENGINEER-EMT,FIRE,F,Salary,,,$103350.00,
"ABBATE, TERRY M",POLICE OFFICER,POLICE,F,Salary,,,$93354.00,
"ABBOTT, BETTY L",FOSTER GRANDPARENT,FAMILY & SUPPORT,P,Hourly,20,,,$2.65
"ABDALLAH, ZAID ",POLICE OFFICER,POLICE,F,Salary,,,$84054.00,
"ABDELHADI, ABDALMAHD ",POLICE OFFICER,POLICE,F,Salary,,,$87006.00,
"ABDELLATIF, AREF R",FIREFIGHTER (PER ARBITRATORS AWARD)-PARAMEDIC,FIRE,F,Salary,,,$102228.00,
"ABDELMAJEID, AZIZ ",POLICE OFFICER,POLICE,F,Salary,,,$84054.00,
"ABDOLLAHZADEH, ALI ",FIREFIGHTER/PARAMEDIC,FIRE,F,Salary,,,$91272.00,
"ABDUL-KARIM, MUHAMMAD A",ENGINEERING TECHNICIAN VI,WATER MGMNT,F,Salary,,,$111492.00,
```

- ▶ Let's start by reproducing our 'high salaries grep' from day 1.



- ▶ Let's start by reproducing our 'high salaries grep' from day 1.

---

```
#!/usr/bin/env python
```

```
for line in open("salaries.csv", "r"):
```

```
    if "$" not in line: continue
```

```
    line = line.replace("$", "").strip()
```

```
    # split the line into a list
```

```
    spline = line.split(",")
```

```
    # pull off the salary, as a float
```

```
    if not spline[-2]: continue
```

```
    if float(spline[-2]) > 200000: print(line)
```

---

# Python: Beyond Single Lines

- ▶ Using bash, we were limited in our 'global' view.
- ▶ Though we could sort, we mainly looked at one line at a time.
- ▶ Python lets us store variables and manipulate the entire dataset.\*
  - ▶ In future weeks we'll learn more and more tools for doing this.

---

\*Truth be told, bash allows this too; it's just less fun.

- ▶ What was the expenditure on salaries in the fire department?
- ▶ Ex. 2: modify `ex/a/dept_salaries.py` to print the total, average, and max salaries, and the number of employees.
  - ▶ Use `len()`, `sum()`, and `max()`.

- ▶ What was the expenditure on salaries in the fire department?

---

```
#!/usr/bin/env python
```

```
total = 0
for l in open("salaries.csv"):

    sl = l.strip().split(",")

    if "FIRE" in l and "$" in sl[-2]:
        total += float(sl[-2][1:])

print("Total salaries: ${:.2f}".format(total))
```

---

- ▶ Ex. 2: modify ex/a/dept\_salaries.py to print the total, average, and max salaries, and the number of employees.
  - ▶ Use `len()`, `sum()`, and `max()`.

# JSON, or, dictionaries and lists revisited.

- ▶ Officially stands for JavaScript Object Notation, but now used in many languages.
- ▶ Common format for transmitting formatted data on the internet.
- ▶ Readily manipulable in Python: just dictionaries and lists.
  - ▶ Can be 'nested' dictionaries – much like classes.
  - ▶ Often, data is packaged with metadata, and you have to 'navigate down' to a list of actually useful data.

```
[
  {
    "B16010_041E": "14855",
    "county": "001",
    "NAME": "Adams County, Pennsylvania",
    "state": "42",
    "B16010_001E": "69921"
  },
  {
    "B16010_041E": "322092",
    "county": "003",
    "NAME": "Allegheny County, Pennsylvania",
    "state": "42",
    "B16010_001E": "871951"
  },
  {
    "B16010_041E": "7270",
    "county": "005",
    "NAME": "Armstrong County, Pennsylvania",
    "state": "42",
    "B16010_001E": "49791"
  },
  {
    "B16010_041E": "27698",
    "county": "007",
    "NAME": "Beaver County, Pennsylvania",
    "state": "42",
    "B16010_001E": "122580"
  }
]
```

Sample JSON Objects:  
Dictionaries in Lists

Loading and saving JSON files uses the same open syntax as other files.

- Focus on reading in, for now.

---

```
import requests, json

# we'll cover this in a few weeks.
j = requests.get("...").json()

# writing to a file
with open("narcotics.json", "w") as out:
    out.write(json.dumps(j, indent=2))

# reading a file
with open("narcotics.json") as data:
    narcotics = json.load(data)
```

---

# JSON: Exploring and Accessing Data (1)

Let's explore some JSON data. Open a python prompt in `lectures/03/ex/b/`, and load `narcotics.json`.

- ▶ What are the most common drug offenses (descriptions)?
  - ▶ Use a set comprehension (curly branches) to get the types.
  - ▶ Use a for loop to ask how many of each type there are.

**Gist/Solution**

## JSON: Exploring and Accessing Data (2)

Now load `ex/b/locations.json` and `ex/b/elevations.json`.

These are Google API responses for geolocating the 100 largest cities in the US, and querying their elevations.

- ▶ How many cities are there, above 1 mile?
- ▶ What is the highest “big city” in the US?
- ▶ Again, explore the data using `l.keys()`.
- ▶ You’ll have to find the highest city first, and then match its coordinates.