Lecture 16 (2-20-26)

Python (CSV, JSON)

```
steam = open('input.txt') #opeining file in python
stream.close() #closing the file in python


# `with` statements only keep the file open while you are in that blokc
with open('input.txt') as stream:
     for line in steam:
          #do some stuff
```

---

streams

Files: `for line in open(path)`
Processes: `for line in os.popen(command)`

`zipcodes.py` script example:

```
import os
import re
import sys

import requests

#Functions

def usage(exit_status = 0):
     printf(''' this is the usage ''')

def zipcodes(city: str, state: str) -> None:
     url = f'https://www.zipcodes.com/{state}/' #using fstring without
printing
```

```
    regex = r'/([^/]+)/[A-Z]{2}/([0-9]{5})/"'

    response = requests.get(url) #response returns an object, not a stream
    #response.text would get you what you actually requested

    mathces = re.findall(regex, response.text) # python equivalent of grep

    for town, zipcode in matches:
        if city is None or city == town:
            print(zipcode)

    #example of printing out line
    for line in response.text.splitlines():
        print(line)


#Main execution
def main(arguments = sys.argc[1:]):
    state = 'Indiana'
    city = None

    while arguments:
        argument = arguments.pop(0)
        match argument:
            case '-c': city = arguments.pop(0)
            case '-s' : state = arguments.pop(0)
            case '-h' : usage(0)
            case _ : usage(1) #remember underscore is default

    zipcodes(city, state)

if __name__ == '__main__':
    main()
```

`wikepedia.py` script example

- **API**: application user interface
  - a way to talk to websites programatically
- **JSON:** data structures nested inside of other data structures

```
#!/usr/bin/env python3
```

```python
import pprint
import re
import sys

import requests

# Constants

HEADERS  = {'User-Agent': __name__}
URL      = 'https://en.wikipedia.org/w/api.php'
PARAMS   = {
    'action'  : 'query',
    'list'    : 'search',
    'format'  : 'json',
    'srsearch': sys.argv[1],
}

def get_size(article):
     return article['size']

# Main Execution

def main():
    response = requests.get(URL, params=PARAMS, headers=HEADERS)
    data = response.json() #converts json to dictionary
    articles = data['query']['search']

    #sorting articles by size(ascending order), and only keeping first 5
    #also passing a function to another function here

    # you could also have `key = lambda a: a['size']` when you pass key
    # this is an anonymous function
    articles = sorted(articles, key = get_size, reverse=True)[:5]

    #printing the articles and their size
    for index, article in enumerate(articles, 1):
        title = article['title']
        size = article['size']

        # grap=bbing snipper and replacing HMTL tags with nothing
        #also slicing first 55 characters
        text = re.sub(r'<[^>]+>', '', article['snippet'])[:55]

        if index > 1: #so we only have space in between records
            print()
```

```
        print(f'{index:>4}.\t{title} ({size})\n\t({text})')



if __name__ == '__main__':
    main()
```