THE UNIVERSITY OF CHICAGO

MASTERS IN COMPUTATIONAL SOCIAL SCIENCE
THE UNIVERSITY OF CHICAGO

MACS 30111

Working with Files

Data types recap

- **Lists**: mutable, mix and match types
- **Tuples**: immutable, CAN mix and match types
- **Dictionaries**: key/value pairs
- **Sets**: groupings (present/not), cannot ‘index’

Agenda/Misc

- **Needed file for today:**
- **names:** https://uchicago.box.com/s/kp207rd2vita1a4zz6749oe8jkvk85l6
- **Instructors.txt:** https://uchicago.box.com/s/oim7c3si6p8ju1b72nciqcdgp283ab1f

Topics:

- Basic file I/O
  - Open: load the data from disk
  - Read: manipulate the data
  - Close: print the results or write the data back to disk
- Working with tabular data using CSV files
- Working with JSON files
- Other file formats

Common Programming Pattern

Common pattern when working with data:

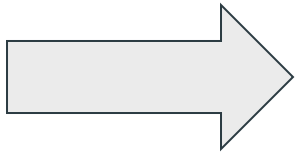
1. **Read** the contents of a file (or files) from disk and **load** the data into one or more data structures
2. **Manipulate** the data in some way
3. **Print** the result or **write** the data back to disk

Sample application

Given a file of email addresses (username@domain), construct a file with the corresponding user names.

`instructor-email.txt`

"email"
"hdambane@uchicago.edu"
"dapeterson@uchicago.edu"
"zwang13@uchicago.edu"
"jclindaniel@uchicago.edu"
"nardin@uchicago.edu"



`instructor-email-short.txt`

Coding practice: 4.1.1

Common Programming Pattern

Common pattern when working with data:

1. **Read the contents of a file (or files) from disk and load the data into one or more data structures**
2. **Manipulate** the data in some way
3. **Print** the result or **write** the data back to disk

Basic File I/O

To access the contents of a file, we first need to open it:

```
f = open("instructor-email.txt")
```

file pointer

To read data from a file, we use the read method:

```
addrs = f.read()
```

read the entire contents into a string

When we are done with a file, we need to close it:

```
f.close()
```

close the file pointer

Coding practice: 4.1.1

Alternative to *close()*

The ***with*** statement to ensure that a file is closed once we're done with it:

```
with open('instructor-email.txt') as f:
    s = f.read()
    email_addresses = sorted(s.split())
```

Coding practice: 4.1.1

Read the file one line at a time

Use a *for* loop to iterate over a text file line by line:

```
with open('instructor-email.txt') as f:
    for line in f:
        print(line)
```

extra empty line

```
with open('instructor-email.txt') as f:
    for line in f.readlines():
        print(line)
```

line.strip()

Coding practice: 4.1.1

Common Programming Pattern

Common pattern when working with data:

1. **Read** the contents of a file (or files) from disk and **load** the data into one or more data structures
2. **Manipulate** the data in some way
3. **Print the result or write the data back to disk**

Write data to a file

To write to a file, we must open the file in **write mode**.

```
with open("names.txt", "w") as f:
    f.write("Anne Rogers\n")
    f.write("Borja Sotomayor\n")
    f.write("Yanjing Li\n")
    f.write("Matthew Wachs\n")
    f.write("Todd Dupont\n")
```

We can also use ***print*** to avoid having to worry about the newline.

```
with open("names2.txt", "w") as f:
    print("Anne Rogers", file=f)
    print("Borja Sotomayor", file=f)
    print("Yanjing Li", file=f)
    print("Matthew Wachs", file=f)
    print("Todd Dupont", file=f)
```

Very important:

- Opening an existing file in write mode will **wipe all its contents**!
- Opening a file that does not exist in write mode will **create** the file.

Coding practice: 4.1.2

Function to trim (can give issues re: writing output)

```
def strip_domain(input_filename, output_filename):
    '''

    Strip the domain names off the email address from the input
    file and write the resulting usernames to the output file.

    Inputs:
      input_filename: (string) name of a file with email addresses
      output_filename: (string) name for the output file.
    '''

    # Load data into a data structure (a list of strings)
    email_addresses = []
    with open(input_filename) as f:
        for line in f:
            email = line.strip()
            email_addresses.append(email)
```

```
#(cont'd)

    # Transform the data
    usernames = []
    for email in email_addresses:
        username, domain = email.split("@")
        usernames.append(username)

    # Write the data
    with open(output_filename, "w") as f:
        for username in usernames:
            print(username, file=f)
```

Summary

The common programming pattern:

1. Load the data from disk:
    a. **Open** a file to **read**
    b. Read the contents of the file from disk
    c. Load the data into a data structure
2. Manipulate the data in some way
3. Print the result or write the data back to disk
    a. **Write** the data
    b. **Close** the file (or use a with statement when you open it)

Coding practice: 4.1.2

Topics:

- Basic file I/O
  - Open: load the data from disk
  - Read: manipulate the data
  - Close: print the results or write the data back to disk
- **Working with tabular data using CSV files**
- Working with JSON files
- Other file formats

CSV (Comma Separated Values) format

CSV files are useful for storing **tabular data**: any data that can be organized into rows, each with the same columns (or "fields")

`instructors.csv`

header

```
id,lname,fname,email
hdambane,Dambanemuya,Henry,hdambane@uchicago.edu
dapeterson,Peterson,David,dapeterson@uchicago.edu
zwang13,Wang,Zhao,zwang13@uchicago.edu
jclindaniel,Clindaniel,Jon,jclindaniel@uchicago.edu
nardin,Nardin,Sabrina,nardin@uchicago.edu
```

What if you want to skip lines?

- Multiple ways to approach:

```
new_lst = []
with open('names.txt') as names:
    next(names)
    next(names)
    next(names)
    new_lst = names.readlines()
```

```
new_list = []
with open('names.txt') as names:
for i, line in enumerate(names):
    if i > 3 and i < 149:
        new_list.append(line)
```

Sample application

1. Read the original data from `instructors.csv`
   - https://uchicago.box.com/s/bay5suooc7nm48gxr0t2cloyfiuzvfb1
2. Manipulate the data by:
   a. getting field information for each row
3. Print the formatted output of the data

Coding practice: 4.1.3

Read file using *csv* module

- *csv.DictReader* - read rows from a CSV file into dictionaries
- *csv.DictWriter* - write dictionaries into rows of a CSV file

Alternatively, we could also use:

- *csv.reader* - read rows from a CSV file into a list of lists
- *csv.writer* - write lists into rows of a CSV file

Different ‘modes’

- You can open files in different ‘modes’
  - r: ‘read’ mode (default)
  - w: ‘write’ mode (needs specified)
  - a: append

If you’re just reading a file, you can operate as normal. If you’re wanting to write a new file, *then* you will use “w”.

DANGER ALERT!!! In “w” mode, you will OVERWRITE THE PREEXISTING FILE!!

Writing files

“w” for “write mode”

```
with open("names_cleaned.txt", "w") as f:
    for build in new_list:
        print(build, file=f)
```

Applied practice!

Working with text

Exercise

- You are working on a project creating a directory of buildings based on where MACSS classes typically meet.
- Use text file: https://uchicago.box.com/s/kp207rd2vita1a4zz6749oe8jkvk85l6

- How would you load your data?
- Next, you want to select the following buildings: sched = ["1155", "SS", "TTI", "K", "STU"]
- and output the new list into a separate file

Bringing the exercise together:

```
# making it pretty:

def get_buildings(input_filename, output_filename, sched):
'''

extract relevant buildings from campus list

Inputs:

input_filename: (string) name of a file with buildings

output_filename: (string) name for the output file.

'''
```

```
# Load data into a data structure (a list of strings)
buildings = []
with open(input_filename) as f:
        for line in f:
                builds = line.strip().split("\t", 1)
                buildings.append(builds)

# Transform the data
buildings_select = []
for line in buildings:
        if line[0] in sched:
                buildings_select.append(line)

# Write the data
with open(output_filename, "w") as f:
        for build in buildings_select:
                print(build, file=f)
```

Exam prep: spot 3 errors and rewrite code

```
# making it pretty:

def get_buildings(input_filename, output_filename, sched):
'''

extract relevant buildings from campus list

Inputs:

input_filename: (string) name of a file with buildings

output_filename: (string) name for the output file.
'''
```

```
# Load data into a data structure (a list of strings)
buildings = []
with open(input_filename) as f:
        for line in f:
                builds = line.strip().split("\t", 1)
                buildings.append(builds)

# Transform the data
buildings_select = []
for line[0] in buildings:
        if line in sched:
                buildings_select.append(line)

# Write the data
with open(output_filename) as f:
        for build in buildings:
                print(build, file=f)
```

Topics:

- Basic file I/O
  - Open: load the data from disk
  - Read: manipulate the data
  - Close: print the results or write the data back to disk
- Working with tabular data using CSV files
- **Working with JSON files**
- Other file formats

JSON (JavaScript Object Notation) format

JSON is a **lightweight** data-interchange / data-storage format commonly used in web services.

Supports different types:

- Object: key-value pairs separated by commas
  - Keys must be strings
  - Values must be valid JSON data types
- Array: empty list or list of objects
- Value: string, number, object, array, true, false, null

File operation using JSON module

String operation:

- *json.dumps*: encodes data into JSON format string
- *json.loads*: decodes JSON format string into a data structure

File operation:

- *json.dump*: encodes data into a JSON file
- *json.load*: decodes data from a JSON file into a data structure

Coding practice: 4.1.4

Topics:

- Basic file I/O
  - Open: load the data from disk
  - Read: manipulate the data
  - Close: print the results or write the data back to disk
- Working with tabular data using CSV files
- Working with JSON files
- **Other file formats**

Other file formats

- HTML: HyperText Markup Language (beautifulsoup)
- XLS, XLSX: Excel formats (xlrd)
- XML: eXtensible Markup Language (beautifulsoup)
- YAML: YAML Ain't Markup Language (yaml)

Application

- You are a social scientist interested in who has won the Nobel Prize.
- In your groups, open up the json data from: https://uchicago.box.com/s/yyq4tf6bxus97ftn08gq0hy85sz13227
- Use section 4.1.4 (near end) to load and inspect the data
- Note: this is a tiny dataset! The goal is to not overwhelm your machine.
- Provide a dictionary of the number of years and awards covered by this dataset.
- **Bonus**! Export this to a json file


Troubleshooting

Troubleshooting code: how do debug

- Break into smaller chunks
- Test each chunk:
  - Are there errors?
  - Does it work as expected?
  - What kind of case might be ‘weird’ … did that work?
- Bring chunks together

Recap

- Sometimes you have text or data that you need to work with
- Python is here for you! You can pull it in and write over it / work with it in files
- Be careful of how you access the text (write may overwrite a file)
- Think about your goals and the best way to work through things

Additional practice

Prep for Thursday!

- Post on ED:
  - One question you have (be specific, include an example)
  - One question you think would be good for the midterm

Optional practice on your own!

- **Data work**: Nobel prizes
  - Pull in data and create a dictionary from an original dataset.
- **Simulation**: Conway's game of life: https://playgameoflife.com/
  - Create a simple simulation of Conway's game of life

Application #1

-
-
-
-