AAE 875 – Fundamentals of Object Oriented Programming and Data Analytics

Cornelia Ilin, PhD

Department of Ag & Applied Economics
UW-Madison

Week 4 - Summer 2019

Chapter 11: More on Lists and Dictionaries

- List nesting
- List comprehensions
- Dictionary nesting
- Functions and Methods

List nesting

```
# print seed prices, in dollars
# 0: GM-IR  1: GM-HT

prices = [
    [
        0,
        56,  # GM-HT
        125 # GM-IR
    ],
    [
        47  # GM-IR
        121  # GM-HT
    ]
]
user_input = input('Enter seed pair (Ex: 0 1) -- ').strip()
seed1, seed2 = user_input.split()
print('Prices: %d dollars' % prices[int(seed1)][int(seed2)])
```

List comprehension

```
# print seed prices, in dollars
# 0: GM-IR  1: GM-HT

prices = [
    [
        0,
        56,  # GM-HT
        125 # GM-IR
    ],
    [
        47  # GM-IR
        121  # GM-HT
    ]
]
sum_list = [sum(row) for row in my_list]
print(sum_list)
```

Dictionary nesting

```
# print seed prices, in dollars

prices = {}
prices = {
    '2018': {
           'GM-IR': [79, 80],
           'GM-HT': [120,  87]
    },
    '2019': {
           'GM-IR': [110, 122],
           'GM-HT': [65, 89]
    }
}
print(prices['2018']['GM-HT'])
```

Functions and Methods

- Covered in the first week (Ch 3)

Chapter 12: IO Files

- Version control
- Reading (Input)
- Writing (Output)
- The 'with' statement
- Interacting with the file systems

Version control

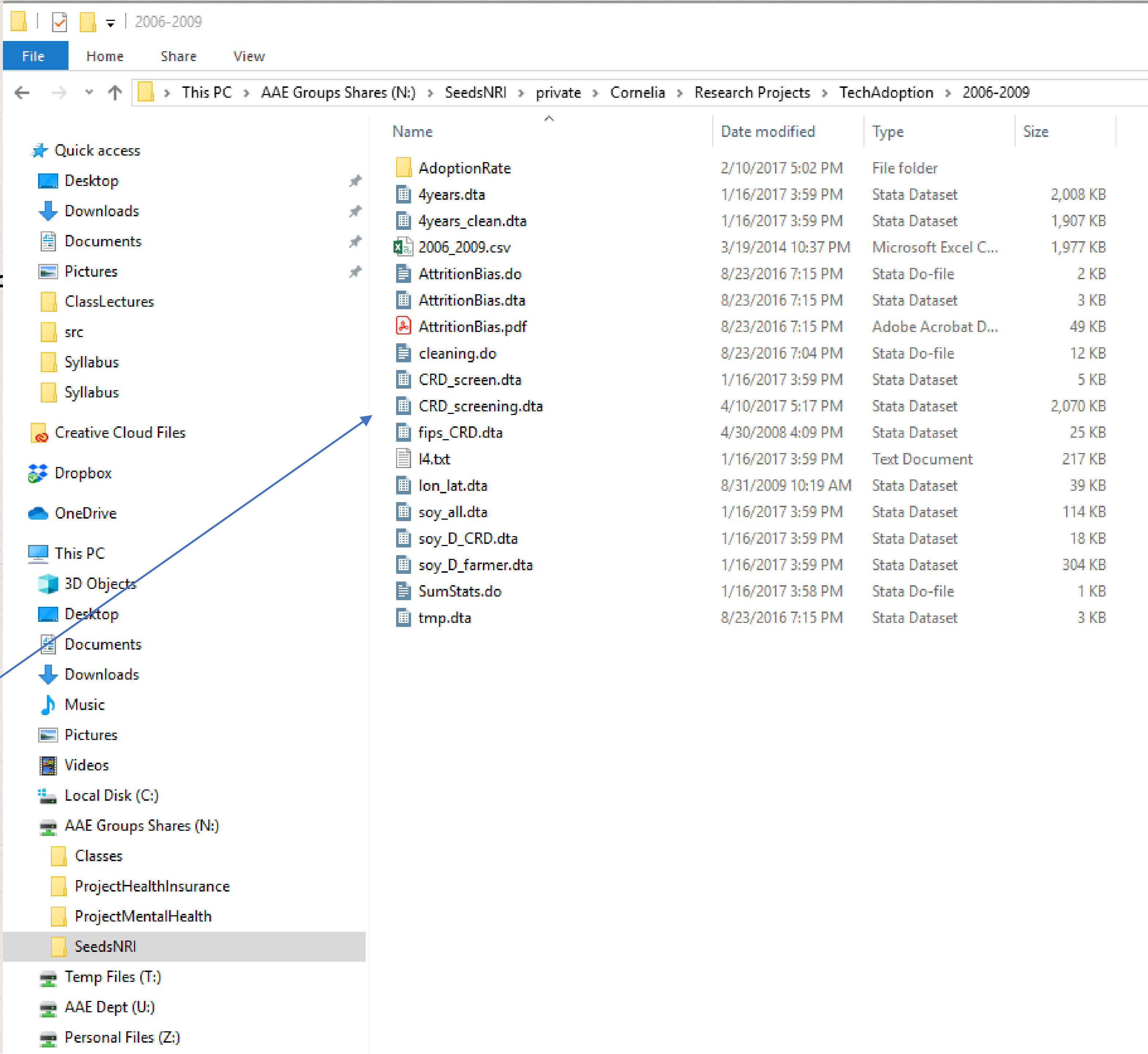
- Extremely important for project file management!

Version control

- Extremely important for project f

This is me in grad school (go ahead and laugh if you feel like!)

Sadly, I am not able to tell you what is the main code, what each data set means… why I have .csv, .txt, .pdf files in the same place with no related meaning… I don't know!!!



2006-2009

File Home Share View

This PC › AAE Groups Shares (N:) › SeedsNRI › private › Cornelia › Research Projects › TechAdoption › 2006-2009

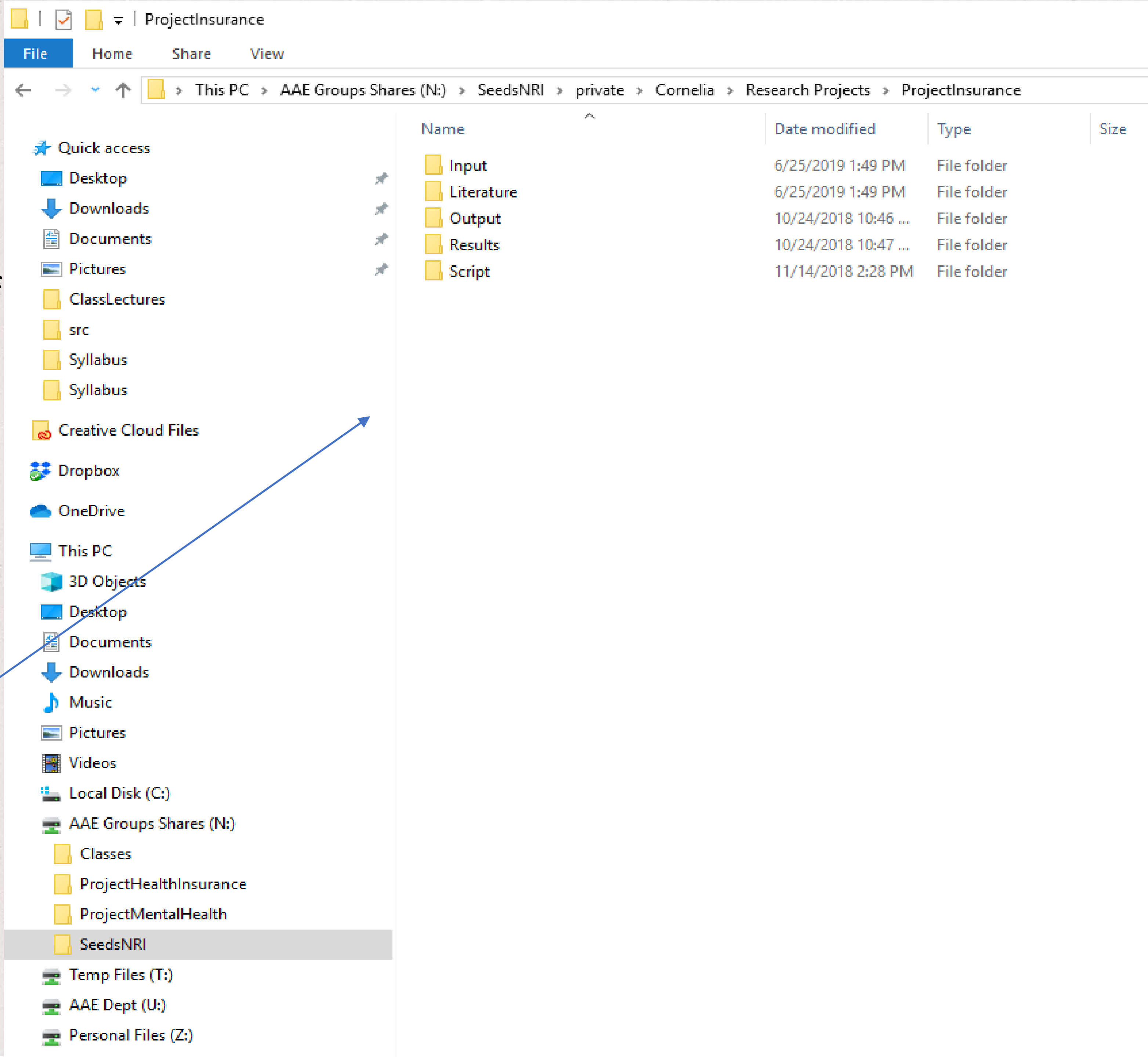
Name	Date modified	Type	Size
AdoptionRate	2/10/2017 5:02 PM	File folder	
4years.dta	1/16/2017 3:59 PM	Stata Dataset	2,008 KB
4years_clean.dta	1/16/2017 3:59 PM	Stata Dataset	1,907 KB
2006_2009.csv	3/19/2014 10:37 PM	Microsoft Excel C...	1,977 KB
AttritionBias.do	8/23/2016 7:15 PM	Stata Do-file	2 KB
AttritionBias.dta	8/23/2016 7:15 PM	Stata Dataset	3 KB
AttritionBias.pdf	8/23/2016 7:15 PM	Adobe Acrobat D...	49 KB
cleaning.do	8/23/2016 7:04 PM	Stata Do-file	12 KB
CRD_screen.dta	1/16/2017 3:59 PM	Stata Dataset	5 KB
CRD_screening.dta	4/10/2017 5:17 PM	Stata Dataset	2,070 KB
fips_CRD.dta	4/30/2008 4:09 PM	Stata Dataset	25 KB
l4.txt	1/16/2017 3:59 PM	Text Document	217 KB
lon_lat.dta	8/31/2009 10:19 AM	Stata Dataset	39 KB
soy_all.dta	1/16/2017 3:59 PM	Stata Dataset	114 KB
soy_D_CRD.dta	1/16/2017 3:59 PM	Stata Dataset	18 KB
soy_D_farmer.dta	1/16/2017 3:59 PM	Stata Dataset	304 KB
SumStats.do	1/16/2017 3:58 PM	Stata Do-file	1 KB
tmp.dta	8/23/2016 7:15 PM	Stata Dataset	3 KB

Version control

- Extremely important for project f

And this is me after grad school

Just ask me a question about this project to see if I know what's going on here ☺



ProjectInsurance

File | Home | Share | View

This PC › AAE Groups Shares (N:) › SeedsNRI › private › Cornelia › Research Projects › ProjectInsurance

Quick access
- Desktop
- Downloads
- Documents
- Pictures
- ClassLectures
- src
- Syllabus
- Syllabus

Creative Cloud Files

Dropbox

OneDrive

This PC
- 3D Objects
- Desktop
- Documents
- Downloads
- Music
- Pictures
- Videos
- Local Disk (C:)
- AAE Groups Shares (N:)
  - Classes
  - ProjectHealthInsurance
  - ProjectMentalHealth
  - SeedsNRI
- Temp Files (T:)
- AAE Dept (U:)
- Personal Files (Z:)

Name	Date modified	Type	Size
Input	6/25/2019 1:49 PM	File folder	
Literature	6/25/2019 1:49 PM	File folder	
Output	10/24/2018 10:46 ...	File folder	
Results	10/24/2018 10:47 ...	File folder	
Script	11/14/2018 2:28 PM	File folder	

Version control

- Extremely important for project file management!
- Allows for collaborations (what if a team of 10 people work on the same project at the same time?)

Version control

- Extremely important for project file management!
- Allows for collaborations (what if a team of 10 people work on the same project?)
- Two types of version control:
  - Local, in a computer's hard drive (cat-proof but not disaster-proof)
  - In the cloud, stored on a server (cat-proof and disaster-proof)

Version control - local

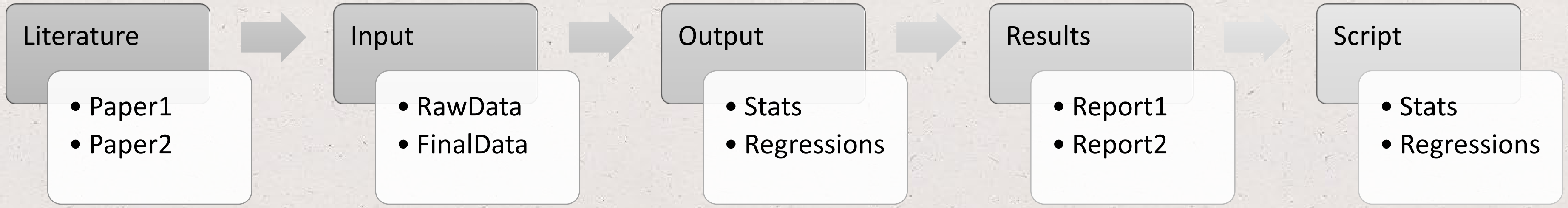
- What if you work for a company whose work is deemed 'highly confidential'?
- You cannot put any work on a server *that* is *located at some address in Western Europe*

Version control - local

- What if you work for a company whose work is deemed '<u>highly confidential</u>'?
- You cannot put any work on a server *that* is *located at some address in Western Europe*
- All you can do is to implement version control in a computer's hard drive
- Allows for collaborations – provided computers are connected to the same network

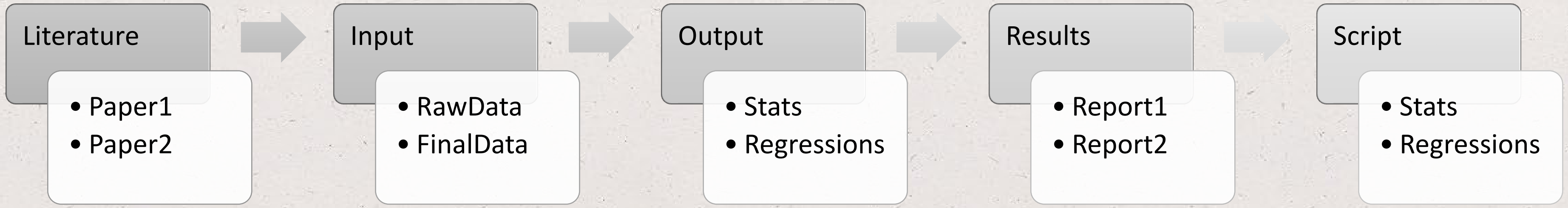
Version control - local

- One example of Project Management (suitable for economists)



Version control - local

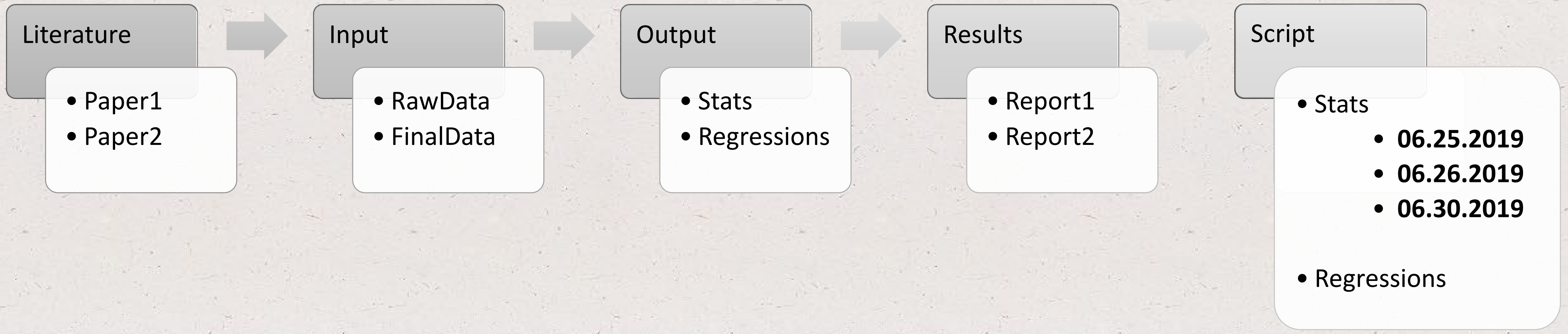
- One example of Project Management (suitable for economists)



- How can we track changes made? (yes, you will change your code multiple times!)

Version control - local

- One example of Project Management (suitable for economists)

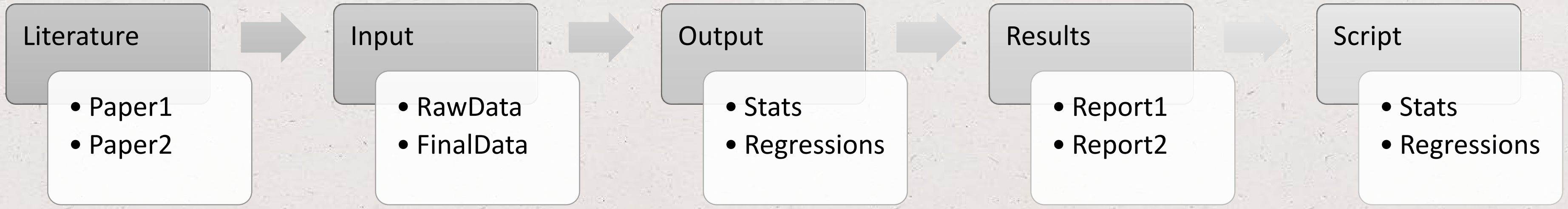


Option 1: Never edit existing code!
Create a new version and edit there

- How can we track changes made? (yes, you will change your code multiple times!)

Version control - local

- One example of Project Management (suitable for economists)



Option 2: use Git, a distributed version-control system
<Topic covered in Lab this week>

- How can we track changes made? (yes, you will change your code multiple times!)

Version control - server

- Code written on your computer's hard drive is cat-proof but not disaster-proof
- Disaster-proof: if your computer explodes there is no way for you to recover the information
- To disaster-proof your work “push” your local project folder to a server (confidentiality?)
- GitHub offers you this service (i.e. GitHub can host a (Git) repository)

Version control - server

- Code written on your computer's hard drive is cat-proof but not disaster-proof
- Disaster-proof: if your computer explodes there is no way for you to recover the information
- To disaster-proof your work “push” your local project folder to a server (confidentiality?)
- GitHub offers you this service (i.e. GitHub can host a (Git) repository)
- Keep in mind that Git ≠ GitHub <covered in Lab this week>

Top Hat Question # 1

Set up a project folder in your own computer. What is the best way to do this if information is confidential?

Data Analytics

- We are just a few steps away from the world of data analytics with Python

Data Analytics

- We are just a few steps away from the world of data analytics with Python

- **What did you learn so far?**
    - Fundamental programming concepts (well, if you can get a data analyst job w/o good command of these concepts let us know!)

Data Analytics

- We are just a few steps away from the world of data analytics with Python

- **What did you learn so far?**
  - Fundamental programming concepts (well, if you can get a data analyst job w/o good command of these concepts let us know!)

  - Data types in Python (compare this with R)

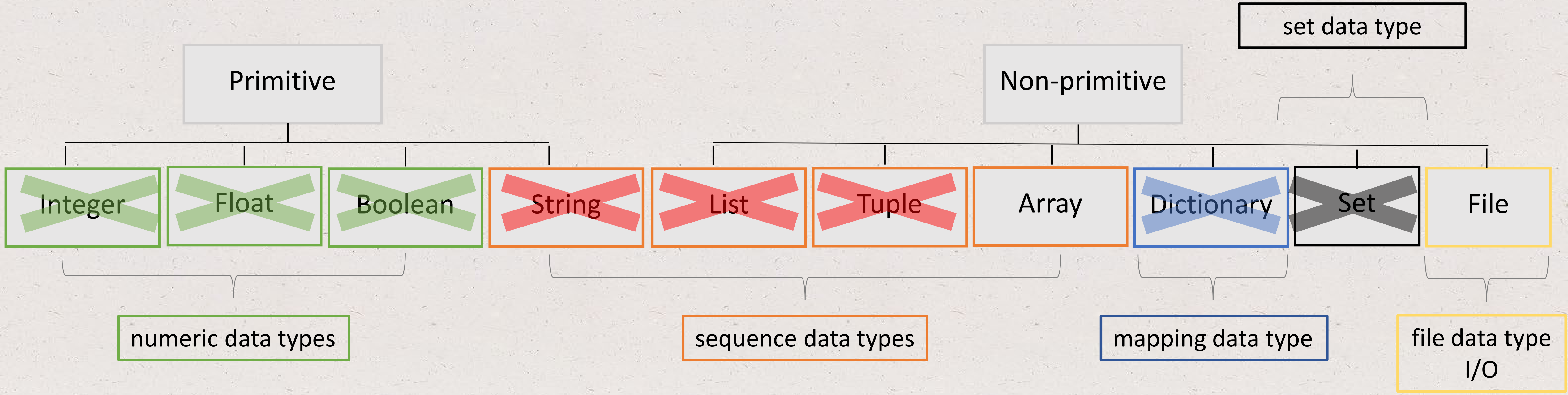
Data Analytics

- We are just a few steps away from the world of data analytics with Python
- **What did you learn so far?**
    - Fundamental programming concepts (well, if you can get a data analyst job w/o good command of these concepts let us know!)
    - Data types in Python (compare this with R)
    - Code/Folder organization

Data Analytics

- We are just a few steps away from the world of data analytics with Python
- **What else do we need to know?**
  - How to input/output files in Python (aka IO files)
  - Data structures in Python

Data types



Primitive
Non-primitive
set data type
Integer
Float
Boolean
String
List
Tuple
Array
Dictionary
Set
File
numeric data types
sequence data types
mapping data type
file data type I/O

Input data

- As economists we often need to read data from a file
- We then need to process that data to produce some useful statistics, regression results, etc.
- Data can come in different forms
- For example data can be in **string** form**, numeric** form and/or **comma separated**
- How to we read it in Python?

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open('workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

More information here:

https://docs.python.org/3.3/tutorial/inputoutput.html

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

The open() function creates a **file object**
The file object is named *stringfile* in this example

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

The open() function creates a **file object**
Most commonly used with two arguments:

- The first argument contains the file name

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

The open() function creates a **file object**
Most commonly used with two arguments:

- The first argument contains the file name
- The second argument (optional) defines the mode the file will be used:

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

The open() function creates a **file object**
Most commonly used with two arguments:

- The first argument contains the file name
- The second argument (optional) defines the mode the file will be used:
    - r: if only reading (this is the default)
    - w: if only writing
    - a: opens the file for appending (data is added to the end)
    - r+, w+, a+ : opens the data for both reading and writing

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

The read() method saves the content of the file object (*stringfile*) as a string

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

The read() method saves the content of the file object (*stringfile*) as a string
Size is an optional argument:

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

The read() method saves the content of the file object (*stringfile*) as a string

Size is an optional argument:

- If omitted or negative, the entire data of the file will be read
- If positive, reads up to ?? bytes

Input data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open(workfile' [, 'w'])

# read the file
data = stringfile.read([size])

# close the file
stringfile.close()
```

← Closes the **file object** (recommended, to save memory)

Input data – string form

- Print the data in the file 'text.txt'

```
# open the file
stringfile = open('text.txt')

# read the file
data = stringfile.read()

# close the file
stringfile.close()

# print the data
print(data)
```

Input data – string form

- Read all the lines of the 'text.txt' file in a **list**. Print only the first line of the file

```
# open the file
stringfile = open('text.txt')

# read the file
data = stringfile.readlines()

# close the file
stringfile.close()

# print the data
print(data[0])
```

The readlines() method returns a list of strings
See also the readline() method

Top Hat Question # 2

What is the output?

```
# open the file
stringfile = open('text.txt')

# read the file
data = stringfile.readlines()

# close the file
stringfile.close()

# print the data
print(data[1])
```

Input data – string form

- Read all the lines of the 'text.txt' file in a **list**. Print the data

```
# open the file
stringfile = open('text.txt')

# read the file
data = stringfile.readlines()

# close the file
stringfile.close()

# print the data
for rows in data:
    print(rows)
```

Top Hat Question # 3

What is the output?

```
# open the file
stringfile = open('text.txt')

# read the file
data = stringfile.readlines()

# close the file
stringfile.close()

# print the data
for rows in stringfile:
    print(rows)
```

Input data – numeric form

- Read all the lines of the 'numeric.txt' file in a **list**. Print the average number

```
# open the file
numfile= open('numeric.txt')

# read the file
data = numfile.readlines()

# close the file
numfile.close()

# compute and print the average
total = 0
for row in data:
    total += int(row)

average = total/len(data)
print('The average is', average)
```

Input data – comma separated form

- Most often data is organized in a spreadsheet format or database (columns x rows)
- A .CSV file separates data items by comma (cells)
- How do we read such data in Python?

Input data – comma separated form

- Most often data is organized in a spreadsheet format or database (columns x rows)
- A .CSV file separates data items by comma (cells)
- How do we read such data in Python?
- The Python **csv module** implements classes to read tabular data in CSV format

```
import csv

# open the file
csvfile = open('workfile' [,'w', newline = ''])

# read the file
data = csv.reader(filename [,delimiter = ','])

# close the file
csvfile.close()
```

More information here:

https://docs.python.org/3/library/csv.html

Input data – comma separated form

- Most often data is organized in a spreadsheet format or database (columns x rows)
- A .CSV file separates data items by comma
- How do we read such data in Python?
- The Python **csv module** implements classes to read tabular data in CSV format

```
import csv

# open the file
csvfile = open('workfile' [,'w', newline = ''])

# read the file
data = csv.reader(filename [,delimiter = ','])

# close the file
csvfile.close()
```

The reader() function in the csv module returns a **reader object** which will iterate over lines in a given .csv file;

Input data – comma separated form

- Most often data is organized in a spreadsheet format or database (columns x rows)
- A .CSV file separates data items by comma
- How do we read such data in Python?
- The Python **csv module** implements classes to read tabular data in CSV format

```
import csv

# open the file
csvfile = open('workfile' [,'w', newline = ''])

# read the file
data = csv.reader(filename [,delimiter = ','])

# close the file
csvfile.close()
```

The reader() function in the csv module returns a **reader object** which will iterate over lines in a given .csv file;

Each row read from the csv file is returned as a **list of strings**;

Input data – comma separated form

- Most often data is organized in a spreadsheet format or database (columns x rows)
- A .CSV file separates data items by comma
- How do we read such data in Python?
- The Python **csv module** implements classes to read tabular data in CSV format

```
import csv

# open the file
csvfile = open('workfile' [,'w', newline = ''])

# read the file
data = csv.reader(filename [,delimiter = ','])

# close the file
csvfile.close()
```

The reader() function in the csv module returns a **reader object** which will iterate over lines in a given .csv file;

Each row read from the csv file is returned as a **list of strings**;

A couple of notes on arguments:

- *filename* is a file object created via open (e.g. csvfile)
- *delimiter* (optional) specifies the argument used in the csv file to separate fields. The default is comma (new cell).

Input data – comma separated form

- Read each row of the 'seeds.csv' file

```
import csv
# open the file
csvfile= open('seeds.csv', 'r+', newline = '')

# read the file
data = csv.reader(csvfile, delimiter = ',')

# print each row
row_no = 1
for row in data:
    print('Row #', row_no, ':', row)
    row_no += 1

# close the file
csvfile.close()
```

Top Hat Question # 4

What is the output?

```
import csv
# open the file
csvfile = open('seeds.csv', 'r+', newline = '')

# read the file
data = csv.reader(csvfile, delimiter = ',')

# print each row
row_no = 1
for row in data:
    print('Row #', row_no, ':', row)

# close the file
csvfile.close()
```

Input data – comma separated form

- Read each row of the 'seeds.csv' file. Compute the average of GM prices paid by farmers

```
import csv
# open the file
csvfile = open('seeds.csv', 'r+', newline = '')

# read the file
data = csv.reader(csvfile, delimiter = ',')

# print average GM price
total_GMprice = 0
row_num = 1
csvfile.readline() # skips the first row in the csv file
```

```
# cont'd
for row in data:
    if row[1] == 'Conv':
        continue
        total_GMprice += int(row[2])
        row_num += 1

mean_GMprice = total_GMprice / row_num
print(mean_GMprice)

# close the file
csvfile.close()
```

Top Hat Question # 5

What is the output?

```
import csv
# open the file
Csvfile = open('seeds.csv', 'r+', newline = '')

# read the file
data = csv.reader(csvfile, delimiter = ',')

# print average GM price
total_GMprice = 0
row_num = 1
csvfile.readline()
csvfile.readline()
```

```
# cont'd

for row in data:
    print(row[2])
    print(row_num)

# close the file
csvfile.close()
```

Top Hat Question # 6

What is the output?

```
import csv
# open the file
csvfile = open('seeds.csv', 'r+', newline = '')

# read the file
data = csv.reader(csvfile, delimiter = ',')

# close the file
csvfile.close()

# print average GM price
total_GMprice = 0
row_num = 1
csvfile.readline()
csvfile.readline()
```

```
# cont'd

for row in data:
    row_num += 1
    print(row[2])
    print(row_num)
```

Top Hat Question # 7

Compute the average quantity sold of GM seeds? Write code. Use the 'seeds.csv' file

Output data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open('workfile' [,'w'])

# write the file
stringfile.write([size])

# close the file
stringfile.close()
```

Output data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open('workfile' [,'w'])

# write the file
stringfile.write([size])

# close the file
stringfile.close()
```

The write() method writes a string to a file

Output data

- Before going into details, 3 commands are useful:

```
# open the file
stringfile = open('workfile' [,'w'])

# write the file
stringfile.write([size])

# close the file
stringfile.close()
```

The write() method writes a string to a file
Integers and floating-points must be converted using the str() function

Output data

- Output 'Hello World' to a file named “my_output.txt”

```
# open the file
stringfile = open('my_output.txt', 'w')

# write the file
stringfile.write('Hello world \n')

# close the file
stringfile.close()
```

Top Hat Question # 8

What is the output?

```
# open the file
stringfile = open('my_output.txt', 'w', newline = '')

# write the file
stringfile.write('Hello world \n')
stringfile.write('I know Python \n')

# close the file
stringfile.close()
```

Top Hat Question # 9

What is the output?

```
# open the file
stringfile = open('my_output.txt', 'w')

# write the file
stringfile.write('Hello world')
stringfile.write('I know Python')

# close the file
stringfile.close()
```

Output data – comma separated form

- How do we output data in comma separated format in Python?
- The Python **csv module** implements classes to output tabular data in CSV format

```
import csv
# open the file
csvfile = open('workfile' [, 'w', newline = ''])

# write the file
data = csv.writer(filename [, delimiter = ' '])
data.writerow('string1'))
data.writerows(['string1', 'string2'])

# close the file
csvfile.close()
```

More information here:

https://docs.python.org/3/library/csv.html

Output data – comma separated form

- How do we output data in comma separated format in Python?
- The Python **csv module** implements classes to output tabular data in CSV format

```
import csv
# open the file
csvfile = open('workfile' [, 'w', newline = ''])

# write the file
data = csv.writer(filename [, delimiter = ' '])
data.writerow('string1'))
data.writerows(['string1', 'string2'])

# close the file
csvfile.close()
```

The writer() function in the csv module returns a **writer object** responsible for converting the user's data into delimited strings on the given workfile

Output data – comma separated form

- How do we output data in comma separated format in Python?
- The Python **csv module** implements classes to output tabular data in CSV format

```
import csv
# open the file
csvfile = open('workfile' [, 'w', newline = ''])

# write the file
data = csv.writer(filename [, delimiter = ' '])
string1 = “Hello”
string2 = “world”
data.writerow(string1))
data.writerows([string1, string2])

# close the file
csvfile.close()
```

The writer functions writerow() and writerows() can be used to write a **list of strings** into the file as one or more rows

Top Hat Question # 10

Add a new row ['2', 'Conv', '55', '10', '2018'**] to the file seeds.csv**

The 'with' statement: Example 1

- What if we forget to close a file? There is a solution to make sure Python **automatically closes the file**: the 'with' statement. It is also a more efficient way to write code

w/o “with”

```
# open the file
stringfile = open('text.txt')

# read the file
data = stringfile.read()

# close the file
stringfile.close()

# print the data
print(data)
```

w/ 'with' version

```
# open the file
with open('text.txt') as stringfile:
    # read the file
    data = stringfile.read()
    # print the data
    print(data)
```

The 'with' statement: Example 2

- What if we forget to close a file? There is a solution to make sure Python **automatically closes the file**: the 'with' statement. It is also a more efficient way to write code

w/o “with” + write()

```
# open the file
stringfile = open('text.txt', 'a+')

# write to the file:
stringfile.write('Let's see if this works')

# read the file
data = stringfile.read()

# close the file
stringfile.close()
```

w/ 'with' version + write()

```
# open the file
with open('text.txt', 'a+”) as stringfile:
    # write to the file
    stringfile.write('Let's see if this works')
    # read the file
    data = stringfile.read()
```

The 'with' statement: Example 3

- What if we forget to close a file? There is a solution to make sure Python **automatically closes the file**: the 'with' statement. It is also a more efficient way to write code

w/o “with” + csv.writer()

```
import csv

# open the file
csvfile = open('seeds.csv', 'a+', newline = '')

# write to the file
data = csv.writer(csvfile)
data.writerow (['2', 'Conv', '55', '10', '2018'])

# close the file
csvfile.close()
```

w/ 'with' version + csv.writer()

```
import csv

# open the file
with open('seeds.csv', 'a+', newline = '') as csvfile:

    # write to the file
    data = csv.writer(csvfile)
    data.writerow(['2', 'Conv', '55', '10', '2018'])
```

Interacting with the file system

- Python comes with the **OS module** that allows your programs to interact with the files in your computer

Interacting with the file system

- Python comes with the **OS module** that allows your programs to interact with the files in your computer

- **File systems:** The computer drive is organized in a hierarchical structure of files and directories
    - *Files*: contain information (e.g. txt, csv files)
    - *Directories*: these contain files and directories inside of them

Interacting with the file system

- Python comes with the **OS module** that allows your programs to interact with the files in your computer

- **File systems:** The computer drive is organized in a hierarchical structure of files and directories
    - *Files*: contain information (e.g. txt, csv files)
    - *Directories*: these contain files and directories inside of them

- **Absolute and relative file paths**:
    - Absolute file paths are notated by a leading forward slash or **drive label**. Describes how to access a given file or directory starting from the root of the file system

      In Windows: Z:\AAE875\EclipseWorkspace\ReadFiles\src

Interacting with the file system

- Python comes with the **OS module** that allows your programs to interact with the files in your computer

- **File systems:** The computer drive is organized in a hierarchical structure of files and directories
  - *Files*: contain information (e.g. txt, csv files)
  - *Directories*: these contain files and directories inside of them

- **Absolute and relative file paths**:
  - Absolute file paths are notated by a leading forward slash or drive label. Describes how to access a given file or directory starting from the root of the file system

    In Windows: Z:\AAE875\EclipseWorkspace\ReadFiles\src

  - Relative file paths are notated by a lack of leading forward slash. A relative file path is interpreted from the perspective of your current working directory (cwd)

    In Windows: src

Interacting with the file system

- Python comes with the **OS module** that allows your programs to interact with the files in your computer

- Why is this important?
  - **Find/change** the **current** working **directory**
  - **Create/remove** files
  - Code **portability** across machines (Windows vs Mac paths)
  - To get the **size of a file**
  - **etc**

Interacting with the file system

- What is the current working directory in Python?

```
import os

print(os.getcwd())
```

```
Z:\AAE875\EclipseWorkspace\ReadFiles\src
```

- How can you change the current working directory to Z:\AAE875\EclipseWorkspace\ReadFiles

```
import os

path = “Z:\AAE875\EclipseWorkspace\ReadFiles”
os.chdir(path)
print(os.getcwd())
```

Top Hat Question # 11

What is the CWD after the following code is run

```
import os

path = “Z:\AAE875\”
os.chdir(path)
print(os.getcwd())
```

Interacting with the file system

- How can you create another directory? CWD is 'Z:\AAE875\EclipseWorkspace\ReadFiles\src'

```
import os
# creates tempDir in current (src) directory
os.mkdir('tempDir')
# creates tempDir in another (ReadFiles) directory
os.mkdir('Z:\AAE875\EclipseWorkspace\ReadFiles\tempDir')
```

relative path

absolute path

- How can you delete *tempDir* located in the src folder?

```
import os
# delete tempDir in current (src) directory
os.rmdir('tempDir')
```

Top Hat Question # 11

What happens in the computer file system when the following code is run?

```
import os
import datetime

curr_day = datetime.datetime.today()

year = str(curr_day.year)
month = str(curr_day.month)
day = str(curr_day.day)
dot = '.'

os.chdir("Z:\AAE875\Eclipse Workspace")
print(os.getcwd())
dir = year + dot + month + dot + day
os.mkdir(dir)
```

Interacting with the file system

- How can you delete files (broadly speaking) in Python
  - os.remove(path) will remove a file
  - os.rmdir(path) will remove an empty directory
  - os.rmtree(path) will remove a directory and all its contents
- **Note:** once you run code with these commands the files are gone! Set up a Test directory!

Interacting with the file system

- How can we make sure the same path is compatible on both Windows and Mac?

```
import os

print(os.getcwd())
```

In Windows:

```
Z:\AAE875\EclipseWorkspace\ReadFiles\src
```

Interacting with the file system

- How can we make sure the same path is compatible on both Windows and Mac?

```
import os

print(os.getcwd())
```

In Windows:

```
Z:\AAE875\EclipseWorkspace\ReadFiles\src
```

```
import os

print(os.getcwd())
```

In Mac:

```
home/AAE875/EclipseWorkspace/ReadFiles/src
```

Interacting with the file system

- How can we make sure the same path is compatible on both Windows and Mac?

In Windows:

```
Z:\AAE875\EclipseWorkspace\ReadFiles\src
```

In Mac:

```
/home/AAE875/EclipseWorkspace/ReadFiles/src
```

- The os.path module contains functions for handling file paths

```
import os

path = os.path.join('Z:\\', 'AAE875', 'EclipseWorkspace', 'ReadFiles', 'src')
```

This command will create a Windows like path if run on a Windows machine