WORKING WITH EXCEL, WORD AND PDF

CS 3030: Python

Instructor: Damià Fuentes Escoté

UCCS University of Colorado Colorado Springs

WORKING WITH EXCEL SPREADSHEETS

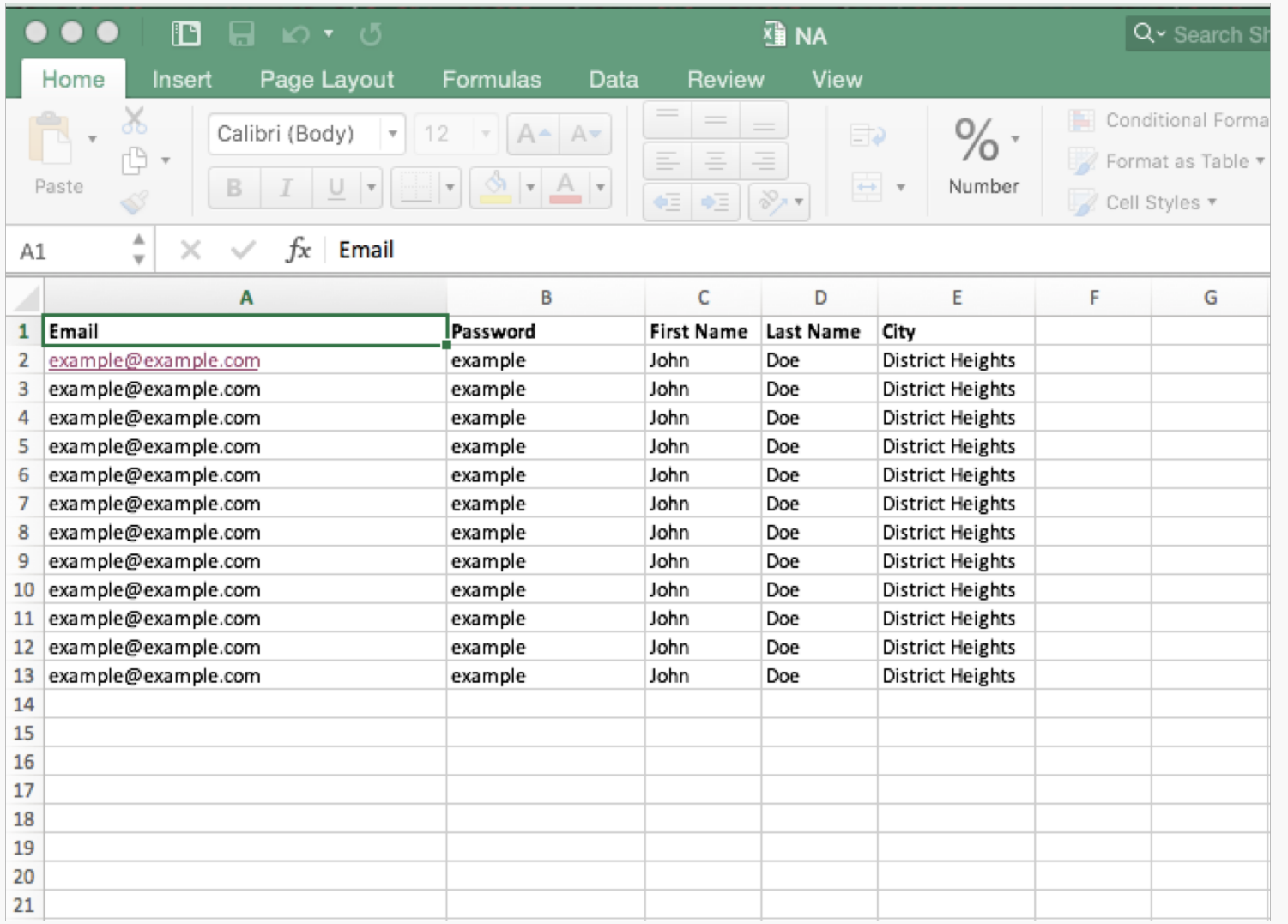
Excel spreadsheets and Python

- Excel is a popular and powerful spreadsheet application.

- The openpyxl module allows your Python programs to read and modify Excel spreadsheet files.
  - *pip install openpyxl*
  - *import openpyxl*
  - *Full documentation >>> http://openpyxl.readthedocs.org/.*

- Although Excel is proprietary software from Microsoft, there are free alternatives that run on Windows, OS X, and Linux that work with Excel's .xlsx file format for spreadsheets:
  - *LibreOffice Calc >>> https://www.libreoffice.org/*
  - *OpenOffice Calc >>> http://www.openoffice.org/*
  - *So, openpyxl module can work on spreadsheets from these applications as well.*

Excel spreadsheet



	A	B	C	D	E	F	G
1	**Email**	**Password**	**First Name**	**Last Name**	**City**		
2	example@example.com	example	John	Doe	District Heights		
3	example@example.com	example	John	Doe	District Heights		
4	example@example.com	example	John	Doe	District Heights		
5	example@example.com	example	John	Doe	District Heights		
6	example@example.com	example	John	Doe	District Heights		
7	example@example.com	example	John	Doe	District Heights		
8	example@example.com	example	John	Doe	District Heights		
9	example@example.com	example	John	Doe	District Heights		
10	example@example.com	example	John	Doe	District Heights		
11	example@example.com	example	John	Doe	District Heights		
12	example@example.com	example	John	Doe	District Heights		
13	example@example.com	example	John	Doe	District Heights		
14							
15							
16							
17							
18							
19							
20							
21							

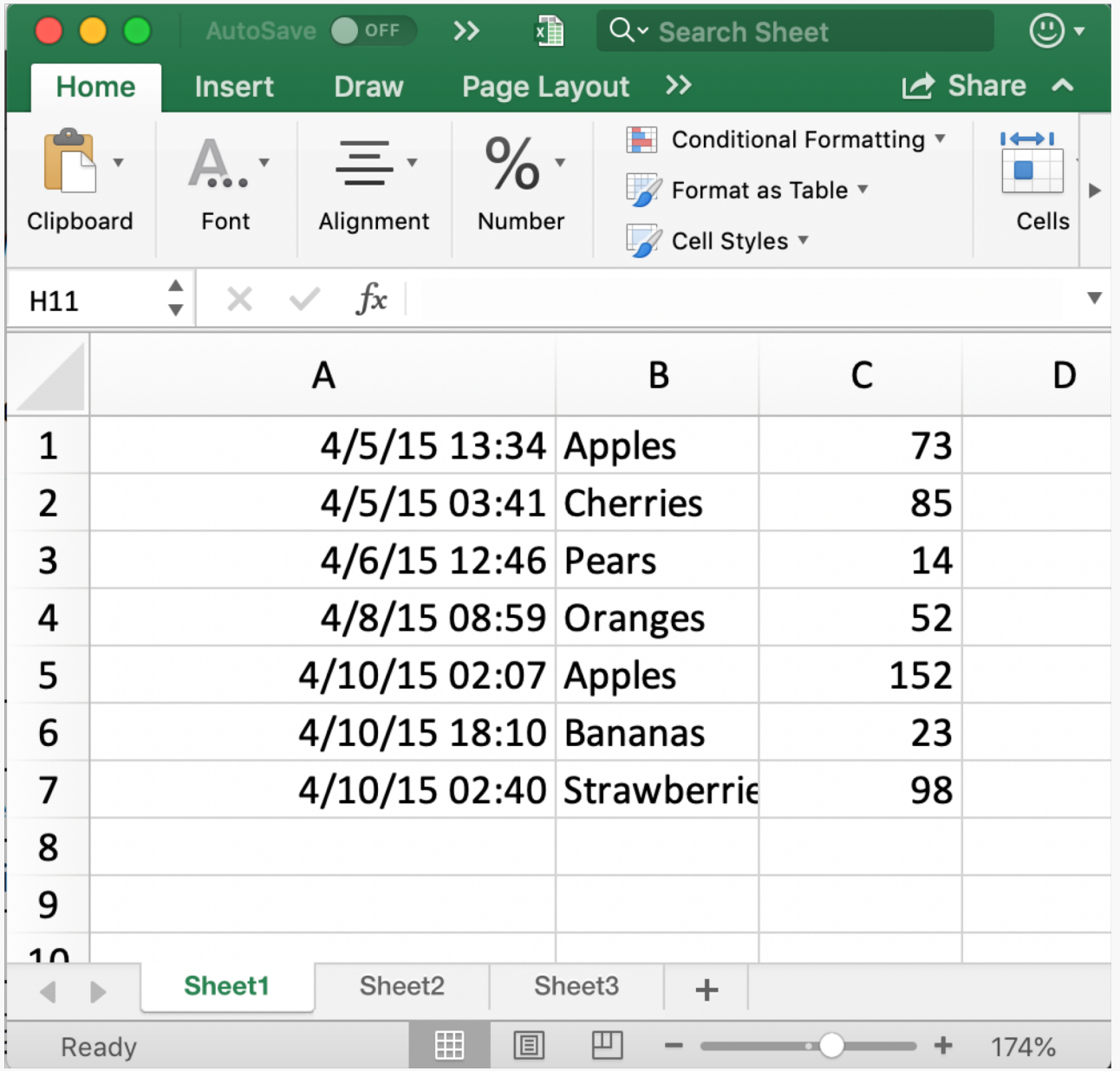
Excel spreadsheets and Python - Examples

- Examples of boring tasks that you could automate with Python:
  - *Copying certain data from one spreadsheet and pasting it into another one.*
  - *Go through thousands of rows and pick out just a handful of them to make small edits based on some criteria.*
  - *Look through hundreds of spreadsheets of department budgets, searching for any that are in the red.*

Definitions

- An Excel spreadsheet document is called a **workbook**.
- A single workbook is saved in a file with the **.xlsx extension**.
- Each workbook can contain multiple sheets (also called **worksheets**).
- The sheet the user is currently viewing (or last viewed before closing Excel) is called the **active sheet**.
- Each sheet has **columns** (addressed by letters starting at A) and rows (addressed by numbers starting at 1).
- A box at a particular column and row is called a **cell**. Each cell can contain a number or text value.
- The grid of cells with data makes up a **sheet**.

example.xlsx



Opening Excel Documents with OpenPyXL

```
import openpyxl


# Remember that example.xlsx needs to be in the current
# working directory in order for you to work with it.
wb = openpyxl.load_workbook('example.xlsx’)
print(type(wb))
# <class 'openpyxl.workbook.workbook.Workbook'>
```

Getting Sheets from the Workbook

```
import openpyxl

wb = openpyxl.load_workbook('example.xlsx’)
print(wb.get_sheet_names())
# ['Sheet1', 'Sheet2', 'Sheet3’]

sheet = wb.get_sheet_by_name('Sheet3’)
print(sheet)
# <Worksheet "Sheet3">
print(type(sheet))
# <class 'openpyxl.worksheet.worksheet.Worksheet’>
print(sheet.title)
# 'Sheet3’

anotherSheet = wb.active
print(anotherSheet)
# <Worksheet "Sheet1">
```

Getting Cells from the Sheets

```
import openpyxl

wb = openpyxl.load_workbook('example.xlsx’)
sheet = wb.get_sheet_by_name('Sheet1’)

print(sheet['A1’])                      # <Cell Sheet1.A1>
print(sheet['A1'].value)                # 2015-04-05 13:34:02
print(type(sheet['A1'].value))          # <class 'datetime.datetime'>
c = sheet['B1’]
print('Row ' + str(c.row) + ', Column ‘ + str(c.column) + ' is ' +
str(c.value))                           # 'Row 1, Column 2 is Apples’
print('Cell ' + c.coordinate + ' is ' + c.value)
                                        # 'Cell B1 is Apples’
```

Converting Between Column Letters and Numbers

```
from openpyxl.utils import get_column_letter,
column_index_from_string


print(get_column_letter(1))                 # 'A’
print(get_column_letter(27))                # 'AA’
print(get_column_letter(900))               # 'AHP’
print(column_index_from_string('A’))        # 1
print(column_index_from_string('AA’))       # 27
```

Getting Cells from the Sheets

- Note that the first row or column integer is 1, not 0.

```
for i in range(1, 8, 2):
    print(i, sheet.cell(row=i, column=2).value)

# 1 Apples
# 3 Pears
# 5 Apples
# 7 Strawberries
```

Determine the size of the sheet

```
import openpyxl


wb = openpyxl.load_workbook('example.xlsx’)
sheet = wb.get_sheet_by_name('Sheet1’)
print(sheet.max_row)
# 7
print(sheet.max_column)
# 3
```

Getting Rows and Columns from the Sheets

```
import openpyxl


wb = openpyxl.load_workbook('example.xlsx’)
sheet = wb.get_sheet_by_name('Sheet1’)
print(tuple(sheet['A1':'C3’]))
# ((<Cell 'Sheet1'.A1>, <Cell 'Sheet1'.B1>, <Cell 'Sheet1'.C1>),
# (<Cell 'Sheet1'.A2>, <Cell 'Sheet1'.B2>, <Cell 'Sheet1'.C2>),
# (<Cell 'Sheet1'.A3>, <Cell 'Sheet1'.B3>, <Cell 'Sheet1'.C3>))
```

Getting Rows and Columns from the Sheets

```
for rowOfCellObjects in sheet['A1':'C3']:
    for cellObj in rowOfCellObjects:
        print(cellObj.coordinate, cellObj.value)

# A1 2015-04-05 13:34:02
# B1 Apples
# C1 73
# A2 2015-04-05 03:41:23
# B2 Cherries
# C2 85
# A3 2015-04-06 12:46:51
# B3 Pears
# C3 14
```

Getting Rows and Columns from the Sheets

```
wb = openpyxl.load_workbook('example.xlsx’)
sheet = wb.active

print(sheet.columns[1])
# (<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>, <Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)

for cellObj in sheet.columns[1]:
    print(cellObj.value)
# Apples
# Cherries
# Pears
# Oranges
# Apples
# Bananas
# Strawberries
```

Quick review - workflow

1. Import the openpyxl module
2. Call the openpyxl.load_workbook() function to get a Workbook object.
3. Call the get_active_sheet() (or .active) or get_sheet_by_name() workbook method to get a Worksheet object.
4. Use indexing or the cell() sheet method with row and column keyword arguments to get a Cell object.
5. Read the Cell object’s value attribute.

Time to code: Reading Data from a Spreadsheet

- Say you have a spreadsheet of data from the 2010 US Census and you have the boring task of going through its thousands of rows to count both the total population and the number of census tracts for each county.
- Each row represents a single census tract.
- Write a script that can read from the census spreadsheet file and calculate statistics for each county in a matter of seconds.

Time to code: Reading Data from a Spreadsheet

- Steps:
  - *Reads the data from the Excel spreadsheet.*
  - *Counts the number of census tracts in each county.*
  - *Counts the total population of each county.*
  - *Prints the results.*
- This means your code will need to do the following:
  - *Open and read the cells of an Excel document with the openpyxl module.*
  - *Calculate all the tract and population data and store it in a data structure.*
  - *Write the data structure to a text file with the .py extension using the pprint module.*

Similar programs

- Compare data across multiple rows in a spreadsheet.
- Open multiple Excel files and compare data between spreadsheets.
- Check whether a spreadsheet has blank rows or invalid data in any cells and alert the user if it does.
- **Read data from a spreadsheet and use it as the input for your Python programs.**

Writing Excel Documents

- OpenPyXL also provides ways of writing data, meaning that your programs can create and edit spreadsheet files.
- With Python, it’s simple to create spreadsheets with thousands of rows of data.

Creating and Saving Excel Documents

```
import openpyxl


wb = openpyxl.Workbook()
print(wb.get_sheet_names())
# ['Sheet']
sheet = wb.active
print(sheet.title)
# 'Sheet'
sheet.title = 'Spam Bacon Eggs Sheet'
print(wb.get_sheet_names())
# ['Spam Bacon Eggs Sheet']
```

Creating and Saving Excel Documents

- Any time you modify the Workbook object or its sheets and cells, the spreadsheet file will not be saved until you call the save() workbook method
- Whenever you edit a spreadsheet you've loaded from a file, you should always save the new, edited spreadsheet to a different filename than the original. That way, you'll still have the original spreadsheet file to work with in case a bug in your code caused the new, saved file to have incorrect or corrupt data.

Creating and Saving Excel Documents

```
import openpyxl


wb = openpyxl.load_workbook('example.xlsx’)
sheet = wb.active
sheet.title = 'Spam Spam Spam’
wb.save('example_copy.xlsx')
```

Creating and Removing Sheets

- Sheets can be added to and removed from a workbook with the create_sheet() and remove_sheet() methods.

```
wb = openpyxl.Workbook()
print(wb.get_sheet_names())        # ['Sheet’]
wb.create_sheet())
print(wb.get_sheet_names())        # ['Sheet', 'Sheet1’]
wb.create_sheet(index=0, title='First Sheet’))
print(wb.get_sheet_names())        # ['First Sheet', 'Sheet', 'Sheet1’]
wb.create_sheet(index=2, title='Middle Sheet’)
print(wb.get_sheet_names())# ['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

Creating and Removing Sheets

- Sheets can be added to and removed from a workbook with the create_sheet() and remove_sheet() methods.
  - *The remove_sheet() method takes a Worksheet object, not a string of the sheet name*

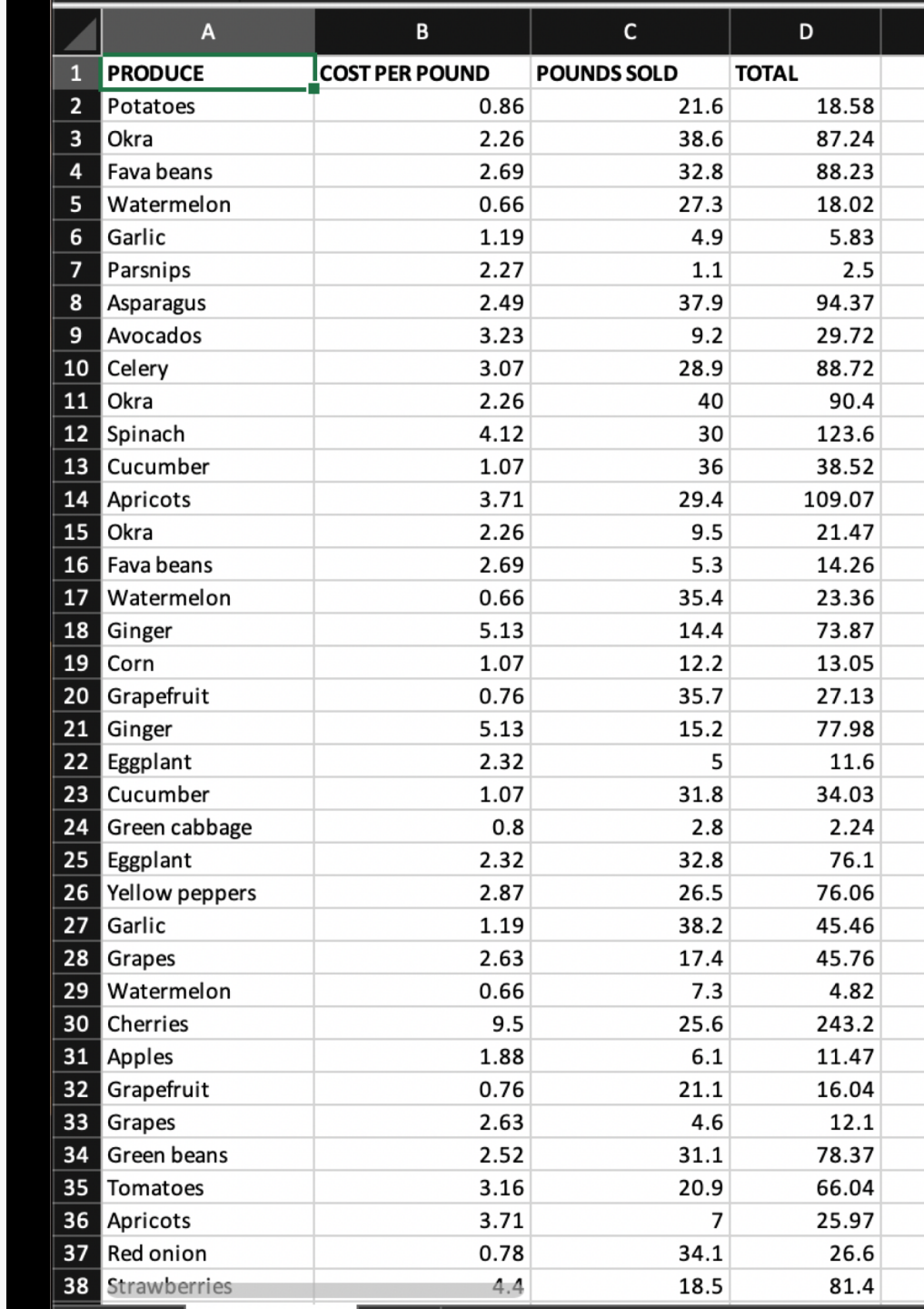
```
print(wb.get_sheet_names())
# ['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1’]
wb.remove_sheet(wb.get_sheet_by_name('Middle Sheet’))
wb.remove_sheet(wb.get_sheet_by_name('Sheet1’))
print(wb.get_sheet_names())   # ['First Sheet', 'Sheet']
```

Writing Values to Cells

```
wb = openpyxl.Workbook()
sheet = wb.get_sheet_by_name('Sheet’)
sheet['A1'] = 'Hello world!’
print(sheet['A1'].value)       # Hello world!
```

Time to code: Updating a spreadsheet

- Imagine you have this product sales spreadsheet and the prices of garlic, celery and lemons were entered incorrectly, leaving you with the boring task of going through thousands of rows in this spreadsheet to update the cost per pound for any garlic, celery, and lemon rows.
- Develop a program that does the following:
  - *Loops over all the rows.*
  - *If the row is for garlic, celery, or lemons, changes the price.*



	A	B	C	D
1	PRODUCE	COST PER POUND	POUNDS SOLD	TOTAL
2	Potatoes	0.86	21.6	18.58
3	Okra	2.26	38.6	87.24
4	Fava beans	2.69	32.8	88.23
5	Watermelon	0.66	27.3	18.02
6	Garlic	1.19	4.9	5.83
7	Parsnips	2.27	1.1	2.5
8	Asparagus	2.49	37.9	94.37
9	Avocados	3.23	9.2	29.72
10	Celery	3.07	28.9	88.72
11	Okra	2.26	40	90.4
12	Spinach	4.12	30	123.6
13	Cucumber	1.07	36	38.52
14	Apricots	3.71	29.4	109.07
15	Okra	2.26	9.5	21.47
16	Fava beans	2.69	5.3	14.26
17	Watermelon	0.66	35.4	23.36
18	Ginger	5.13	14.4	73.87
19	Corn	1.07	12.2	13.05
20	Grapefruit	0.76	35.7	27.13
21	Ginger	5.13	15.2	77.98
22	Eggplant	2.32	5	11.6
23	Cucumber	1.07	31.8	34.03
24	Green cabbage	0.8	2.8	2.24
25	Eggplant	2.32	32.8	76.1
26	Yellow peppers	2.87	26.5	76.06
27	Garlic	1.19	38.2	45.46
28	Grapes	2.63	17.4	45.76
29	Watermelon	0.66	7.3	4.82
30	Cherries	9.5	25.6	243.2
31	Apples	1.88	6.1	11.47
32	Grapefruit	0.76	21.1	16.04
33	Grapes	2.63	4.6	12.1
34	Green beans	2.52	31.1	78.37
35	Tomatoes	3.16	20.9	66.04
36	Apricots	3.71	7	25.97
37	Red onion	0.78	34.1	26.6
38	Strawberries	4.4	18.5	81.4

Setting the Font Style of Cells

- From the Product Sales example, say you want to highlight those items that the cost per pound is greater than $5.

Setting the Font Style of Cells

```
import openpyxl
from openpyxl.styles import Font

wb = openpyxl.Workbook()
sheet = wb.get_sheet_by_name('Sheet')
italic24Font = Font(size=24, italic=True)
sheet['A1'].font = italic24Font
sheet['A1'] = 'Hello world!'
wb.save('styled.xlsx')
```

Setting the Font Style of Cells

```
myfont = Font(size=24, italic=True)
```

Keyword argument	Data type	Description
name	String	The font name, such as 'Calibri' or 'Times New Roman'
size	Integer	The point size
bold	Boolean	True, for bold font
italic	Boolean	True, for italic font

Formulas

- Formulas, which begin with an equal sign, can configure cells to contain values calculated from other cells.
- openpyxl module can add formulas to cells programmatically, just like any normal value.

Formulas

```
wb = openpyxl.Workbook()
sheet = wb.active
sheet['A1'] = 200
sheet['A2'] = 300
sheet['A3'] = '=SUM(A1:A2)’
print(sheet['A3'].value)
# =SUM(A1:A2)
wb.save('writeFormula.xlsx')
```

Formulas

```
wbDataOnly = openpyxl.load_workbook('writeFormula.xlsx',
data_only=True)

sheet = wbDataOnly.activeprint(sheet['A3'].value)

# None or 500
```

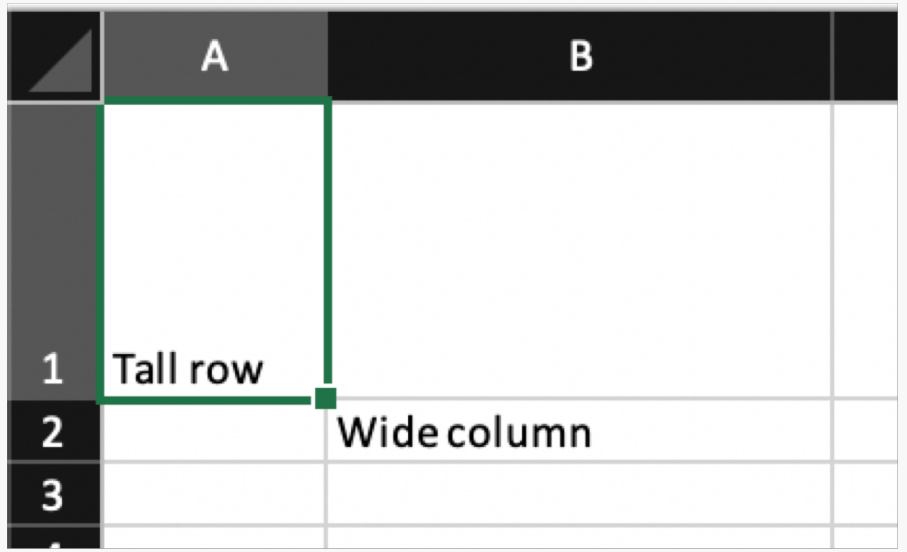
Formulas

```
wbDataOnly = openpyxl.load_workbook('writeFormula.xlsx',
data_only=True)
sheet = wbDataOnly.activeprint(sheet['A3'].value)
# None or 500
# It depends upon the provenance of the file. data_only=True
# depends upon the value of the formula being cached by an
# application like Excel. If, however, the file was created
# by openpyxl or a similar library, then it's probable that
# the formula was never evaluated and, thus, no cached value
# is available and openpyxl will report None as the value
# > > > Try to modify the file with Excel and save it
```

Setting Row Height and Column Width

- Worksheet objects have **row_dimensions** and **column_dimensions** attributes that control row heights and column widths.

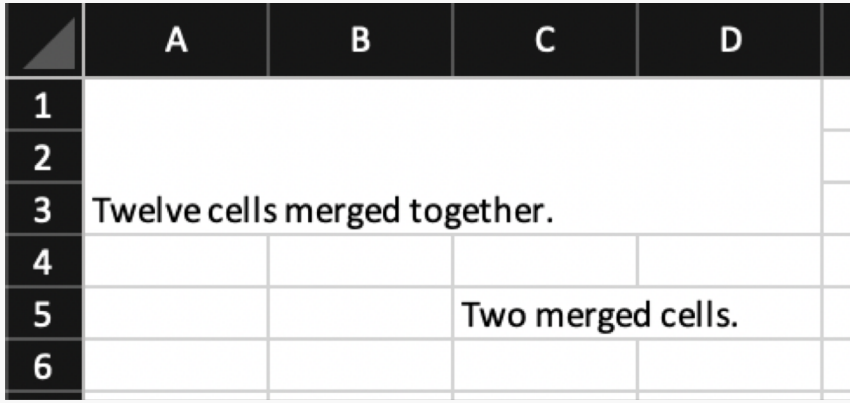
```
wb = openpyxl.Workbook()
sheet = wb.activesheet['A1'] = 'Tall row’
sheet['B2'] = 'Wide column’
sheet.row_dimensions[1].height = 70
sheet.column_dimensions['B'].width = 20
wb.save('dimensions.xlsx')
```



Merging and Unmerging Cells

- A rectangular area of cells can be merged into a single cell with the merge_cells() sheet method
- To unmerge cells, call the unmerge_cells() sheet method.

```
wb = openpyxl.Workbook()
sheet = wb.active
sheet.merge_cells('A1:D3’)
sheet['A1'] = 'Twelve cells merged together.
sheet.merge_cells('C5:D5’)
sheet['C5'] = 'Two merged cells.’
wb.save('merged.xlsx')
```



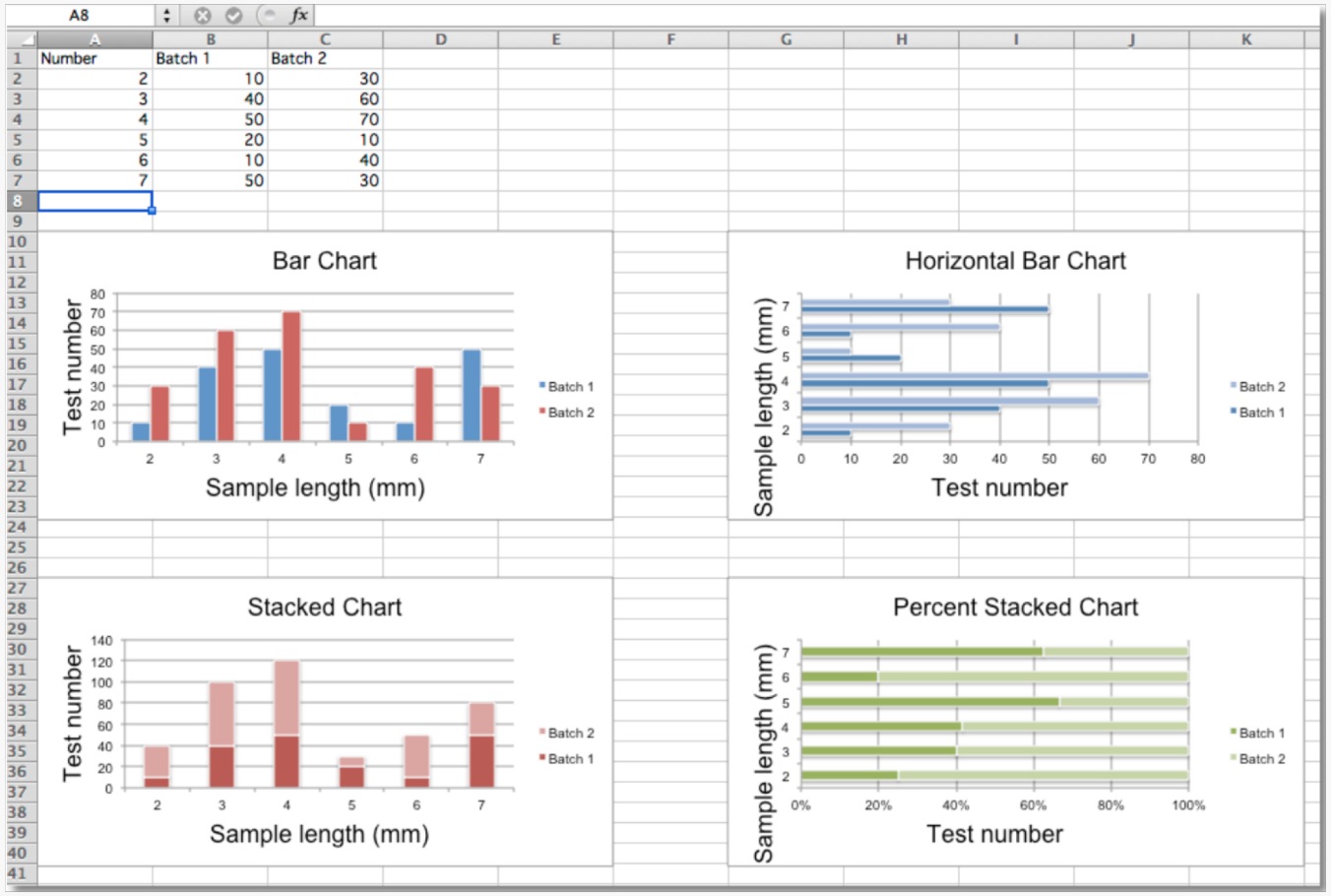
Freeze Panes

- For spreadsheets too large to be displayed all at once, it's helpful to "freeze" a few of the top rows or leftmost columns onscreen. Frozen column or row headers, for example, are always visible to the user even as they scroll through the spreadsheet.
- In OpenPyXL, each Worksheet object has a **freeze_panes** attribute that can be set to a Cell object or a string of a cell's coordinates.
- Note that all rows above and all columns to the left of this cell will be frozen, but the row and column of the cell itself will not be frozen.

```
sheet.freeze_panes = 'C2'
# Columns A and B and row 1 are freezed
```

Charts

- This would be another lesson. So, for more information:
  - *https://openpyxl.readthedocs.io/en/stable/charts/introduction.html*



Time to code – HW 8 ex 1 – Spreadsheet cell inverter

- Write a program to invert the row and column of the cells in the spreadsheet.

	A	B
1	Apples	73
2	Cherries	85
3	Pears	14
4	Oranges	52
5	Apples	152
6	Bananas	23
7	Strawberries	98

	A	B	C	D	E	F	G
1	Apples	Cherries	Pears	Oranges	Apples	Bananas	Strawberries
2	73	85	14	52	152	23	98
3							

WORKING WITH PDF DOCUMENTS

Working with word and pdf documents

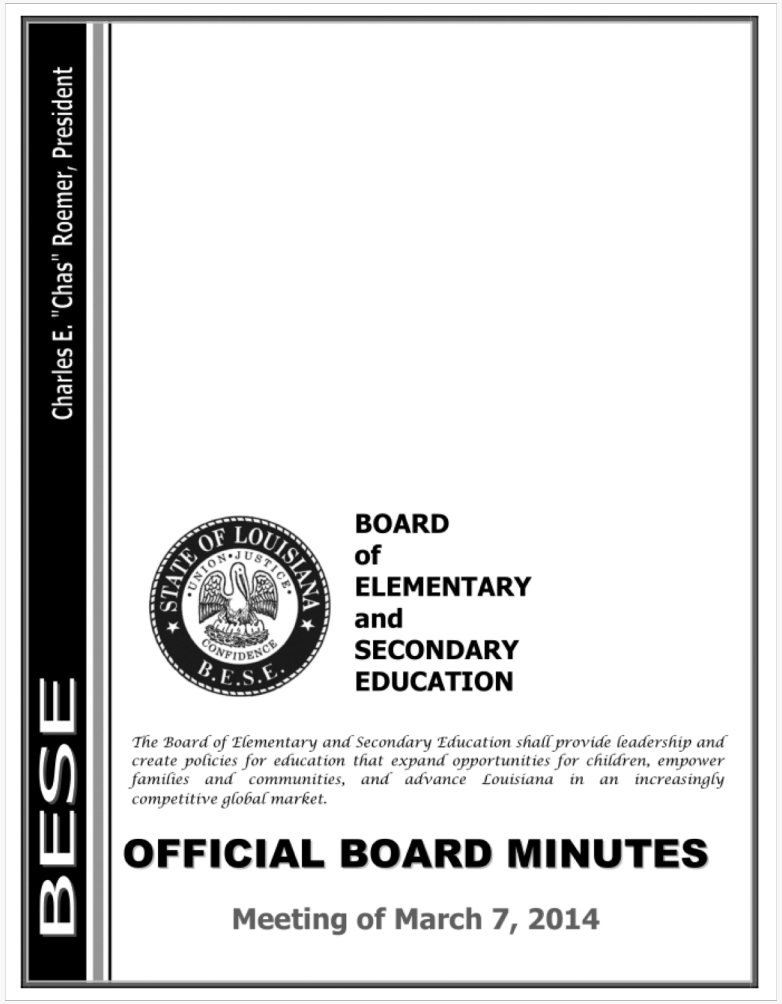
- PDF and Word documents are binary files, which makes them much more complex than plaintext files. In addition to text, they store lots of **font**, **color**, and **layout information**.
- If you want your programs to read or write to PDFs or Word documents, you'll need to do more than simply pass their filenames to open().
- Fortunately, there are Python modules that make it easy for you to interact with PDFs and Word documents:
  - *PyPDF2*
  - *Python-Docx*

PDF Documents

- **PDF** stands for *Portable Document Format* and uses the *.pdf* file extension.
  - *pip install PyPDF2*

- PyPDF2 might make mistakes when extracting text from a PDF and may even be unable to open some PDFs at all. But that is because how PDF are made.
- PyPDF2 does not have a way to extract images, charts, or other media from PDF documents, but it can extract text and return it as a Python string.

Extracting text from PDFs



Extracting text from PDFs

```
import PyPDF2

pdfFileObj = open('meetingminutes.pdf', 'rb')
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
print(pdfReader.numPages)                    # 19
pageObj = pdfReader.getPage(0)
print(pageObj.extractText())
# OOFFFFIICCIIAALL  BBOOAARRDD  MMIINNUUTTEESS   Meeting of
# March 7
# , 2014
#
# The Board of Elementary and Secondary Education shall provide leadership and
# create policies for education that expand opportunities for children, empower
# ...
```

Creating PDFs

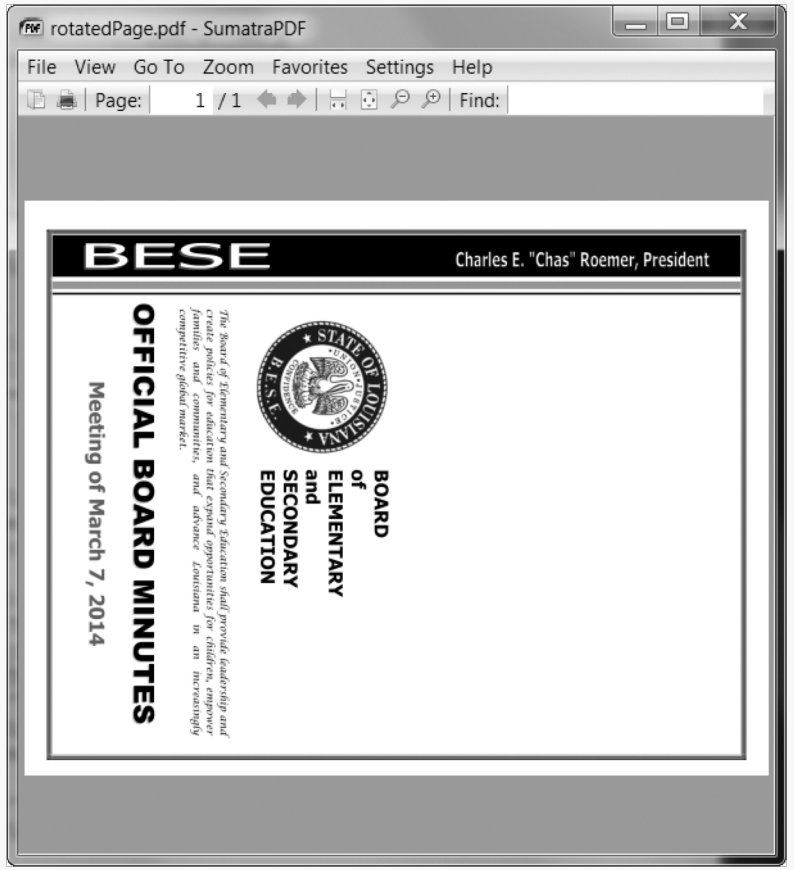
- PyPDF2 cannot write arbitrary text to a PDF like Python can do with plaintext files. Instead, PyPDF2's PDF-writing capabilities are limited to:
  - *Copying pages from other PDFs*
  - *Rotating pages*
  - *Overlaying pages*
  - *Encrypting files*

Copying pages from other PDFs

```
pdf1File = open('meetingminutes.pdf', 'rb')
pdf2File = open('meetingminutes2.pdf', 'rb')
pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
pdfWriter = PyPDF2.PdfFileWriter()

for pageNum in range(pdf1Reader.numPages):
    pageObj = pdf1Reader.getPage(pageNum)
    pdfWriter.addPage(pageObj)

for pageNum in range(pdf2Reader.numPages):
    pageObj = pdf2Reader.getPage(pageNum)
    pdfWriter.addPage(pageObj)

pdfOutputFile = open('combinedminutes.pdf', 'wb')
pdfWriter.write(pdfOutputFile)
pdfOutputFile.close()
pdf1File.close()
pdf2File.close()
```

Rotating Pages

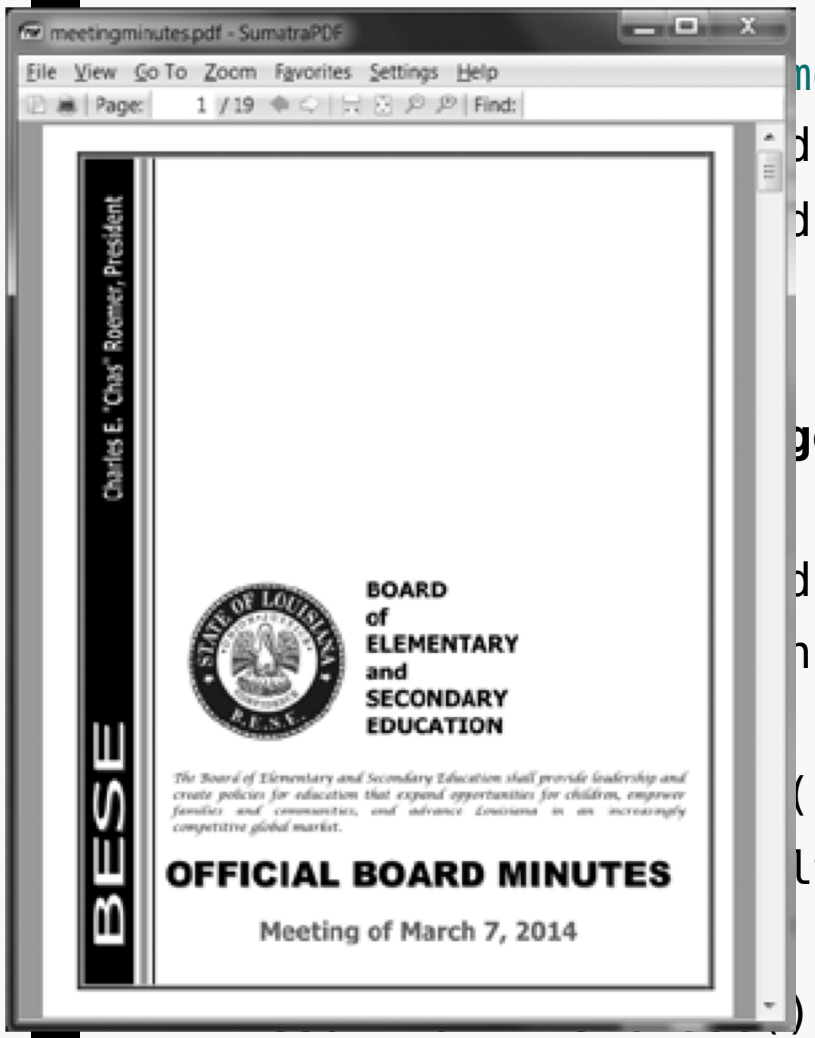
```
minutesFile = open('meetingminutes.pdf', 'rb')
pdfReader = PyPDF2.PdfFileReader(minutesFile)

page = pdfReader.getPage(0)
page.rotateClockwise(90)

pdfWriter = PyPDF2.PdfFileWriter()
pdfWriter.addPage(page)

resultPdfFile = open('rotatedPage.pdf', 'wb')
pdfWriter.write(resultPdfFile)
resultPdfFile.close()
```

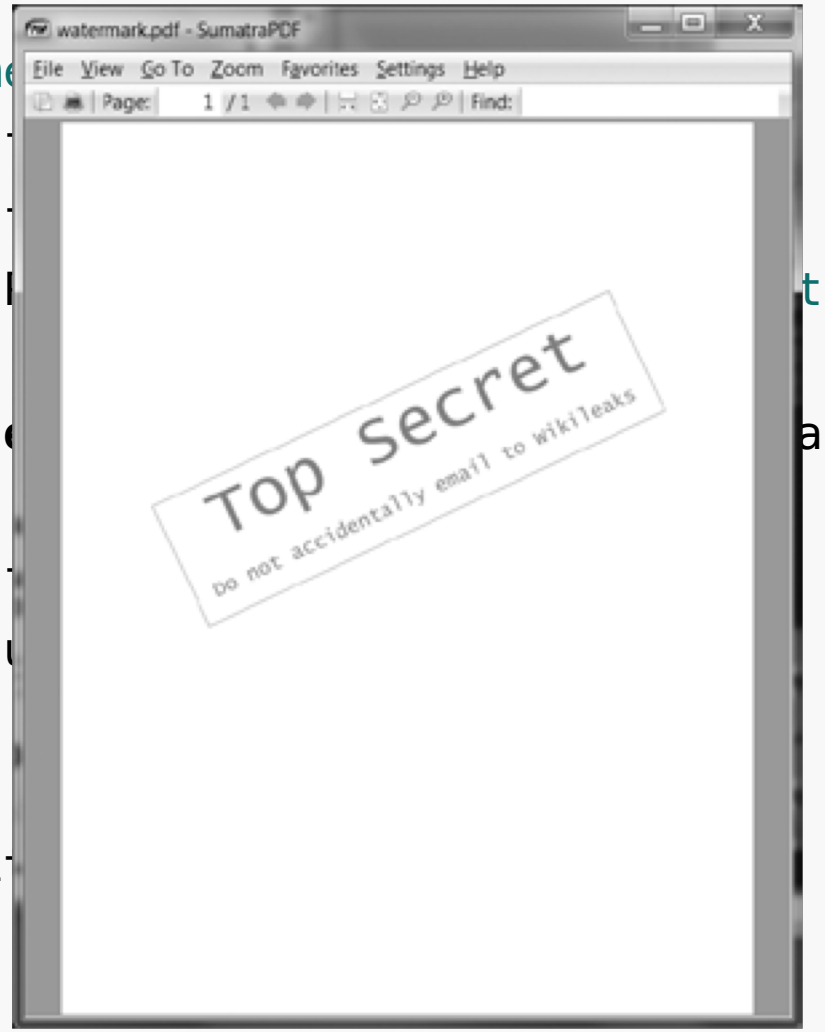


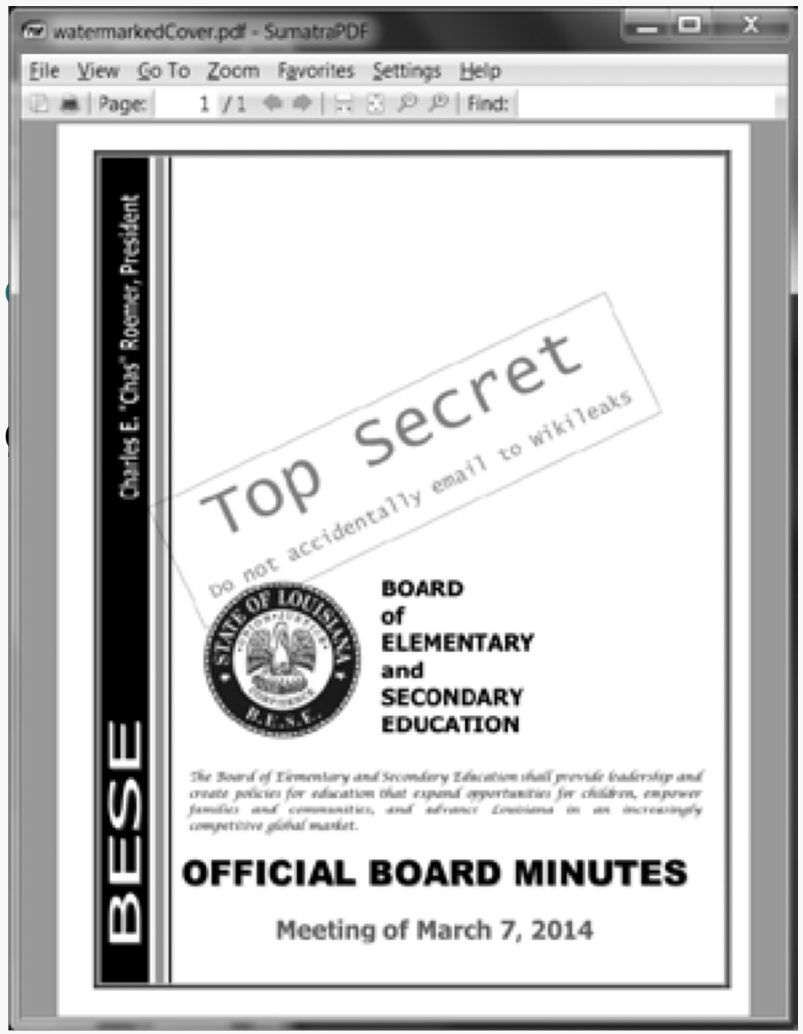
Overlaying Pages

```
minutesFile = open('meetingminutes.pdf', 'rb’)
pdfReader = PyPDF2.PdfFileReader(minutesFile)
minutesFirstPage = pdfReader.getPage(0)
pdfWatermarkReader = PyPDF2.PdfFileReader(open('watermark.pdf', 'rb’))

minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))

pdfWriter = PyPDF2.PdfFileWriter()
pdfWriter.addPage(minutesFirstPage)

resultPdfFile = open('watermarkedCover.pdf', 'wb’)
pdfWriter.write(resultPdfFile)
minutesFile.close()
resultPdfFile.close()
```

Overlaying Pages







Encrypt

```
pdfFile = open('meetingminutes.pdf', 'rb')
pdfReader = PyPDF2.PdfFileReader(pdfFile)
pdfWriter = PyPDF2.PdfFileWriter()

for pageNum in range(pdfReader.numPages):
    pdfWriter.addPage(pdfReader.getPage(pageNum))

pdfWriter.encrypt('swordfish')
resultPdf = open('encryptedminutes.pdf', 'wb')
pdfWriter.write(resultPdf)
resultPdf.close()
```

Decrypt

```
pdfReader = PyPDF2.PdfFileReader(open('encryptedminutes.pdf',
'rb'))
print(pdfReader.isEncrypted)          # True
pdfReader.getPage(0)
# PyPDF2.utils.PdfReadError: file has not been decrypted
pdfReader.decrypt('swordfish')
pageObj = pdfReader.getPage(0)        # Now is working
```

Time to code – HW 8 ex 2 – Brute-force password

- The encrypted.pdf is an encrypted PDF using a single English word. Write a program that will decrypt the PDF by trying every possible English word until it finds one that works. This is called a brute-force password attack.
- Use the dictionary.txt. This dictionary file contains over 44,000 English words with one word per line.
- Loop over each word in this list, passing it to the decrypt() method.
- If your program find the decrypted word, it should break out of the loop and print the hacked password.
- You should try both the uppercase and lowercase form of each word.
- Use the time module to calculate how much time it take to decrypt it.

Time to code – HW 8 ex 2 – Brute-force password

- The encrypted.pdf is an encrypted PDF using a single English word. Write a program that will decrypt the PDF by trying every possible English word until it finds one that works. This is called a brute-force password attack.
- Use the dictionary.txt. This dictionary file contains over 44,000 English words with one word per line.
- Loop over each word in this list, passing it to the decrypt() method.
- If your program find the decrypted word, it should break out of the loop and print the hacked password.
- You should try both the uppercase and lowercase form of each word.
- Use the time module to calculate how much time it take to decrypt it.

You probably want to have the decrypted pdf ;)

WORKING WITH WORD DOCUMENTS

Working with Word documents

- It is as systematic as working with spreadsheets and PDFs.
- pip install python-docx
- The things you can do are:
  - *Read word documents by Run objects. A Run object is a contiguous run of text with the same style.*
  - *Getting the full text from a .docx file*
  - *Styling Paragraph and Run Objects*
  - *Creating Word Documents with Nondefault Styles*
  - *Run Attributes*
  - *Writing Word Documents*
  - *Adding Headings*
  - *Adding Line and Page Breaks*
  - *Adding Pictures*
- For how to do these things go to the book pages 306 to 316.