Spreadsheets and Spreadsheet Files

Spreadsheet programs can read, write data in various formats...

Spreadsheet Files

- Various formats
  - .xls, .xlsx – MS Excel format
  - .ods – Open Document Format
  - .csv – Comma-Separated-Values
  - others as well
- Binary versus human-readable formats
  - .xls, xlsx, .ods formats preserve formatting, other features
    - Generally only usable w/ a spreadsheet program
  - .csv format only saves cell contents (and cell structure)
    - text-only, readable and usable by anything

Organization of Spreadsheet Files

- ***Spreadsheet***: array of rows and columns
  - Intersection of a column and a row is a *cell*
  - Columns identified by letters of the alphabet
    - Column Z is followed by Columns AA, AB, AC, etc.
  - Rows identified by row numbers
    - 1 – 1,048,576 is a common range
  - Cells identified by Column name and Row number
    - A1 is uppermost, leftmost cell
- Spreadsheet files contain ***workbooks***
    - (Workbooks confusingly also called "spreadsheets")
  - Workbook contains 1 or more individual spreadsheets
    - Each spreadsheet has a unique name

Spreadsheet Contents

- Spreadsheet cells contain:
  - Numbers (which are displayed in different formats)
  - Labels – arbitrary text strings
  - Formulas – expressions and functions that operate on other cells
  - (Formatting "meta-information" about contents)
- Spreadsheet files store cell contents
  - Dedicated spreadsheet files: all contents
    - .Excel: **.xls**, .**xlsx**
    - LibreOffice Calc: .**ods**
    - other proprietary formats
  - **.csv**-format files: numbers and labels only

Python and Spreadsheets

- **csv** module supports .csv files
  - Standard
  - Portable to all spreadsheet programs
  - No formatting or formulas; numeric and text contents only
- **openpylx** module supports .xls, .xlsx, .ods, formats
  - Non-standard module, must be added manually
- Other non-standard modules also exist

csv Module

- File must first be opened normally
  - handle = open('filename.csv', 'r')
- Opened file handle is supplied to
  csv.reader() – returns lists of rows
  or
  csv.writer() – writes rows to file
  - Value separator can be specified
    - Defaults to a comma
- For writing out:
  - csv.writer() object provides ".writerow()" method to write out a list as a spreadsheet row

Example: Reading a CSV File

- Open diskfile for reading
- Supply handle to csv.reader()
- Retrieve lists representing spreadsheet rows
- Close diskfile

```
def main(argv=[__name__]):
    import csv

    filename = input('What file? ')
    with open(filename, 'r') as h:
        data = [ lst for lst in csv.reader(h) ]

    print(data)
```

Writing a CSV Diskfile

- Open diskfile for writing
- Supply handle to csv.writer()
- Use csv.writer.writerow() to write lists
  - One list per spreadsheet row
- Close diskfile

```
with open('out.csv', 'w') as h:
    writer = csv.writer(h)
    for lst in data:
        writer.writerow(lst)
```

Exercise – writing .csv

- Build 10 lists of 20 random values
  - np.random.randint()
- Write them into a .csv file
- Inspect file with text editor
- Inspect with spreadsheet program

- Optional:
  - New python code:
    - Read each row from .csv file into list
    - Calculate:
      arithmetic mean, geometric mean, harmonic mean

Exercise – reading .csv

- Browse to https://www.eia.gov/state/ (https://www.eia.gov/state/)
  - Download "download table data as csv"
- Open file with "csv.reader()"
  - Read in rows of data
- Form lists for:
  - state abbrev.
  - production
  - consumption
  - expenditure
- Convert strings to numbers as needed!

Spreadsheets with Column Headers

- 1st row of spreadsheet *labels* the columns
- Actual data in 2nd through last rows
- Treatment:
  - Create a *dictionary* from each data row
  - Dictionary keys formed from labels found in 1st row
  - Collect all dictionaries into a list?
- Works for labels in 1st row
  - Optional: specify dictionary keys with optional "filednames" parameter

DictReader Code Example

- Open diskfile for reading, as before
- Supply handle to csv.DictReader()
- Retrieve a dictionary for each row except 1st
- Close diskfile

```
import csv
filename = input('What input file? ')
with open(filename, 'r') as h:
    dreader = csv.DictReader(h)
    rowdicts = [ ]
    for d in dreader:
        rowdicts.append( d )

print(len(rowdicts), 'data rows in', filename)
for row in rowdicts:
    print(row)
```

Code Example Continued

	A	B	C	D	E
1	ID number	lastname	firstname	lucky number	secret
2	1	Aston	Martin	3.14159	I'm a car
3	2	Bibbly	Brenda	10	Make me up
4	3	Cardamom	Spicy	-99	not very spicy
5	4	Destiny	Manifest	0	evil

```
ID number,lastname,firstname,lucky number,secret
1,Aston,Martin,3.14159,Iâ▒▒m a car
2,Bibbly,Brenda,10,Make me up
3,Cardamom,Spicy,-99,not very spicy
4,Destiny,Manifest,0,evil
```

```
$ python3 dictreader.py
What input file? sample.csv
4 data rows in sample.csv
{'secret': 'I'm a car', 'ID number': '1', 'lucky number': '3.14159', 'firstname'
: 'Martin', 'lastname': 'Aston'}
{'secret': 'Make me up', 'ID number': '2', 'lucky number': '10', 'firstname': 'B
renda', 'lastname': 'Bibbly'}
{'secret': 'not very spicy', 'ID number': '3', 'lucky number': '-99', 'firstname
': 'Spicy', 'lastname': 'Cardamom'}
{'secret': 'evil', 'ID number': '4', 'lucky number': '0', 'firstname': 'Manifest
', 'lastname': 'Destiny'}
$
```

The "openpyxl" module

- Nonstandard Python module
- Opens .xlsx file as a *workbook*
  - Multiple spreadsheets as members of workbook
  - Preserves formatting, formulas, etc.
- Access *sheets* in a workbook
- Access *cells* in a sheet
  - Can use (row, column) notation
    - sheet.**min_row**, sheet.**max_row**
    - sheet.**min_column**, sheet.**max_column**
      - sheet.**max_column** is 1 greater than last column ?
  - Can use "spreadsheet" notation – "A1", "C23", etc.
- Access cell *values* - .value

Example: openpyxl

```
#!/usr/bin/env python3
# 2020-04-29
import pprint
import openpyxl
filename = input('What input file? ')

myWB = openpyxl.load_workbook( filename )
print(myWB.sheetnames)
sheet = myWB.active
print('Home cell:', sheet['A1'] )

# use numeric cell references:
rowdicts = []
for r in range(2, sheet.max_row+1):
    d = {}
    for c in range(1, sheet.max_column):
        key = sheet.cell(row=1, column=c).value
        value = sheet.cell(row=r, column=c).value
        d[key] = value
    rowdicts.append(d)

pprint.pprint(rowdicts)
```

```
What input file? sample.xlsx
['Sheet1']
Home cell: <Cell 'Sheet1'.A1>
[{'ID number': 1,
  'firstname': 'Martin',
  'lastname': 'Aston',
  'lucky number': 3.14159,
  'secret': 'I'm a car'},
 {'ID number': 2,
  'firstname': 'Brenda',
  'lastname': 'Bibbly',
  'lucky number': 10,
  'secret': 'Make me up'},
 {'ID number': 3,
  'firstname': 'Spicy',
  'lastname': 'Cardamom',
  'lucky number': -99,
  'secret': 'not very spicy'},
 {'ID number': 4,
  'firstname': 'Manifest',
  'lastname': 'Destiny',
  'lucky number': 0,
  'secret': 'evil'}]
```

also...

Enumerating Lists

- Common action: accessing list elements according to index:

```
for index in range(len(datalst)):
    value = datalst[index]
    print(index, value)
```

- Alternatively, the "enumerate()" function returns both index and element:

```
for index, value in enumerate(datalst):
    print(index, value)
```

Enumerating Dictionaries

- Obtain *pairs* of key, value from a dictionary
  - Like enumerating a list

```
for rd in rowdicts:
    for key, value in rd.items():
        print(key, value, end='    ')
    print()
```

Introduction to Plotting

- Make list of y-values
- Make corresponding list of x-coordinates

- import matplotlib.pyplot as plt

- plt.plot(x-coordinates, y-values)

  or

- plt.bar(x-coordinates, y-values)

Bar-plotting Example

```
import csv
import matplotlib.pyplot as plt

with open('SelectedStateRankingsData.csv', 'r') as h:
    rdr = csv.reader(h)
    energy = [ row for row in rdr ]

states = [ row[0] for row in energy[1:] ]
production = [ float(row[2]) for row in energy[1:] ]
consumption = [ float(row[4]) for row in energy[1:] ]
expenditures = [ float(row[6]) for row in energy[1:] ]
x_coords = range( len(production) )

plt.bar(x_coords, production, tick_label=states)
plt.show()
```