

SECTION 8: FILE I/O

File I/O

2

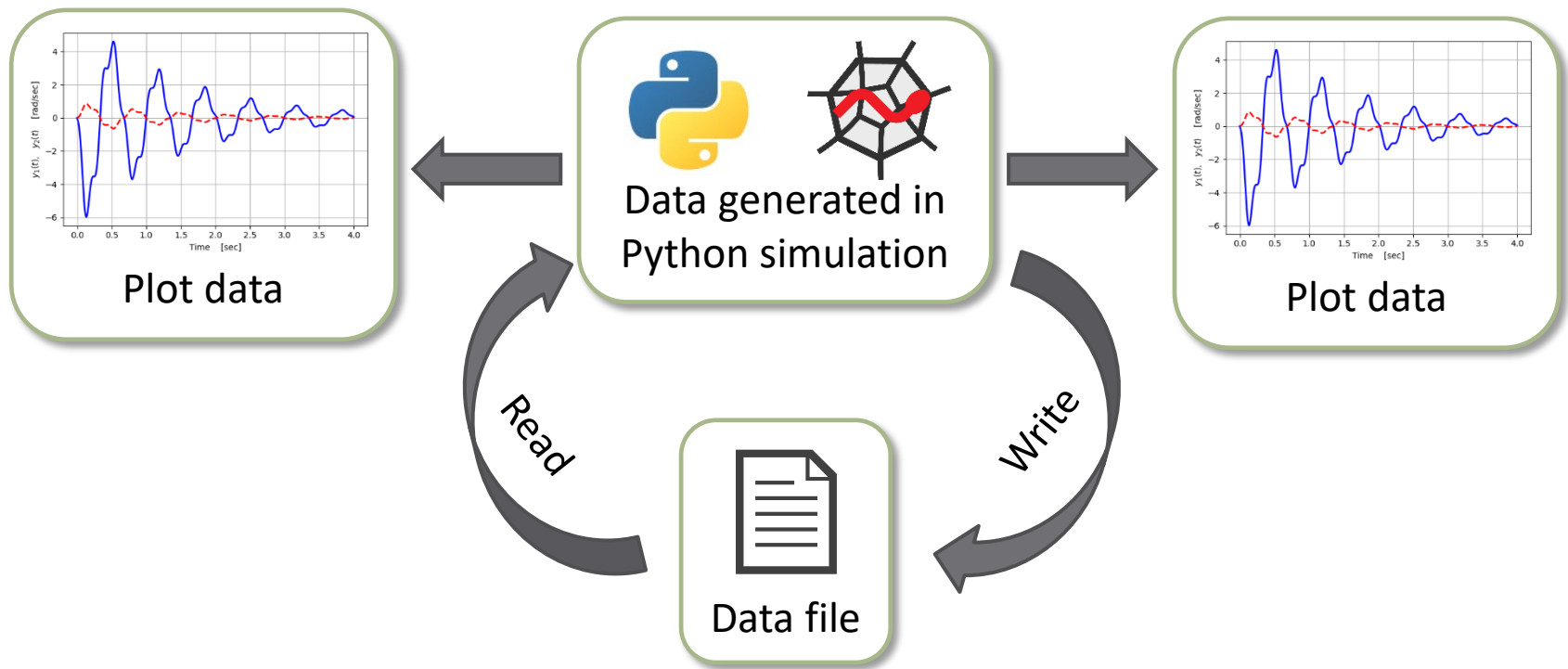
- As engineers, we often generate large amounts of data
 - ▣ Simulation – in Python or other simulation tools
 - ▣ Measurements

- Often need to process and analyze these data
 - ▣ Export data from simulator to a file
 - ▣ Read data using a Python script
 - ▣ Process data using Python – analysis, display, etc.
 - ▣ Write the data generated using Python to a file

File I/O - Examples

3

- We'll go through an example of writing and reading data from a file in several different ways to introduce several of Python's file I/O options:



4

Low-Level File I/O

Low-Level File I/O

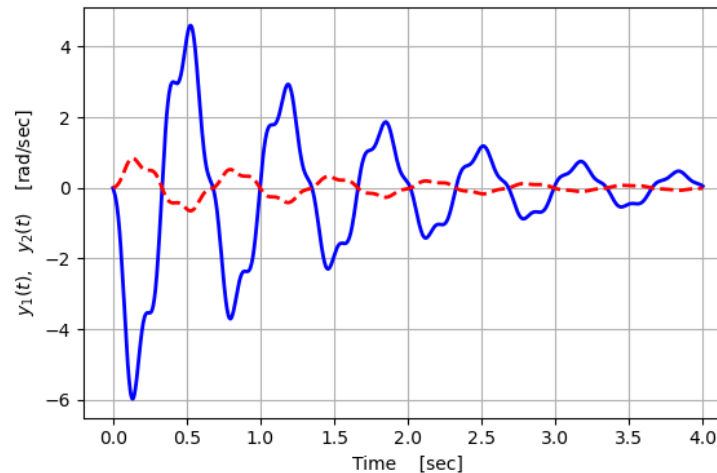
5

- Python packages (e.g. Pandas) includes many ***high-level functions*** for easily importing data from files
 - ▣ Usually use these – very easy to use
 - ▣ Covered later in the notes
- Python also includes ***low-level functions*** for reading from and writing to files
 - ▣ More of a ***manual operation*** – line-by-line operation
 - ▣ ***Similar to other computer languages*** (e.g. C), which may not include simple high-level file I/O functions

File I/O - Example

6

- Let's say we performed a simulation of some sort of dynamic system using Python
- Resulting data set:



- Three data arrays:
 - Time vector, t , and two outputs, $y_1(t)$, $y_2(t)$
- First, we'll use low-level, built-in Python functions to write the data to a file

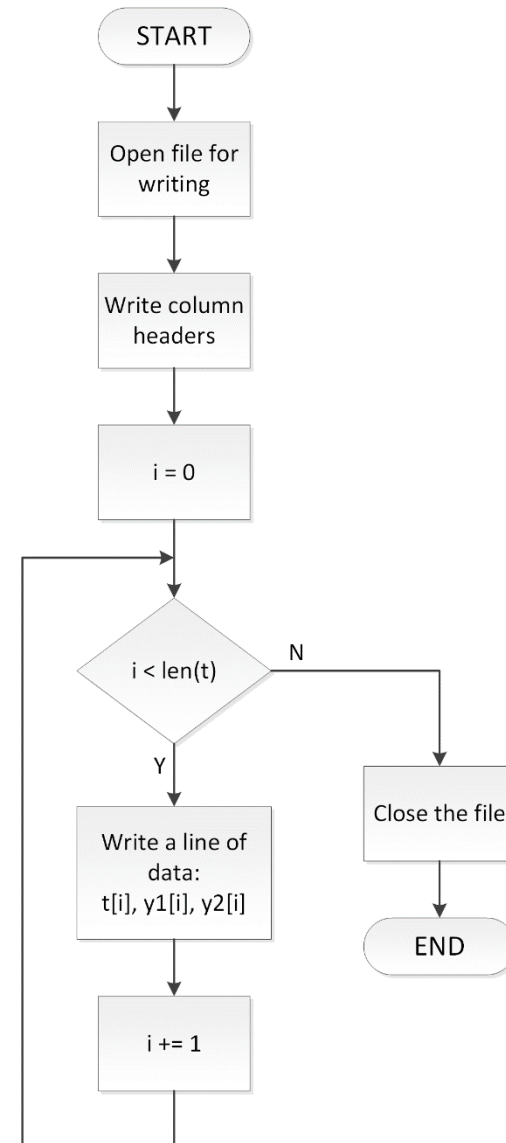
7

Writing to Files

File I/O – Writing

8

- Use low-level, built-in Python functions to write data to a file line-by-line
- The basic procedure:



Opening a Text File – open()

9

- Prior to reading from or writing to a text file, we must first ***open the file***

```
fileObj = open(filename, mode)
```

- `filename`: name of the file to open – need not exist yet – a *string*
- `mode`: *optional* – a string specifying file access type, e.g. read-only, write access, etc. – default is read-only
- `fileObj`: a file object of type `TextIOWrapper` – has associated I/O methods, such as `f.write()` and `f.readlines()`

File Open Modes

10

- Optional mode sequences indicate the type of file access when opening a file

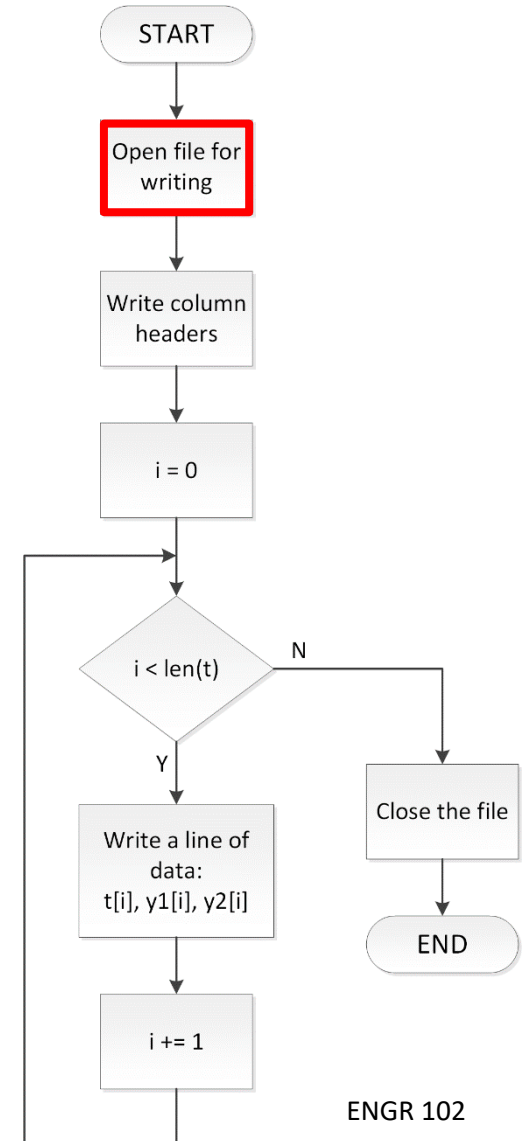
Mode String	Description
'r'	Open file for reading (default)
'w'	Open or create new file for writing – discard existing contents
'a'	Open or create new file for writing – append data to the end of the file
'r+'	Open file for reading and writing

File I/O – Writing – Open File

11

- Open file for writing
 - ▣ mode: 'w'
 - ▣ File created if it does not already exist
 - ▣ f is a TextIOWrapper file object

```
30  
31 f = open('dataFile1.txt', 'w')  
32
```



Write to a File – `f.write()`

12

- Apply the `write()` method to the file object, `f`

```
f.write(s)
```

- `f`: file object of type `TextIOWrapper` – returned from the `open()` function
 - `s`: string to be written to `f`
- Writing occurs character-by-character
 - Newlines, spaces, delimiters (e.g., commas, tabs) must be explicitly included in strings


File I/O – Writing – Headers

13

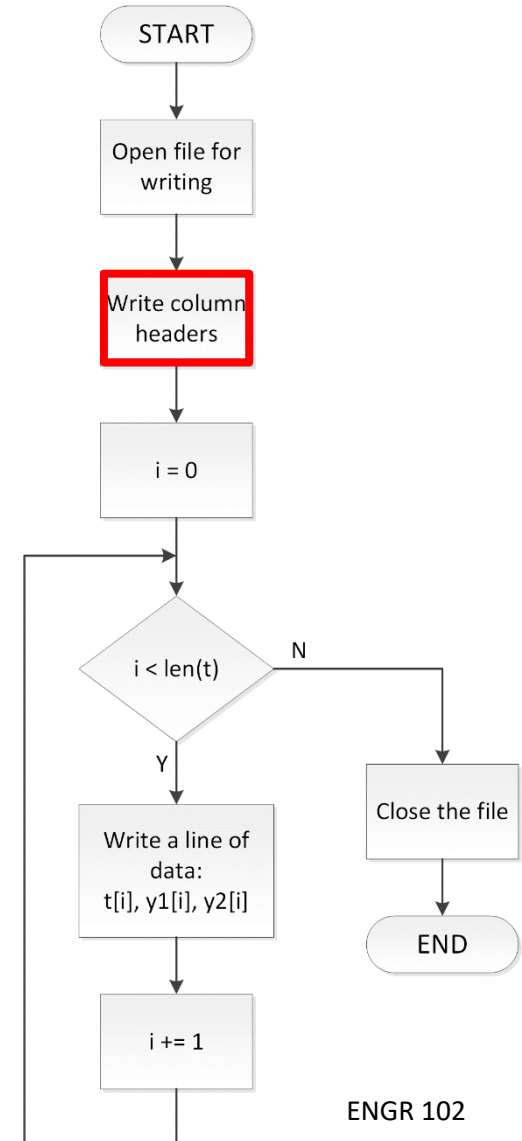
- May want to insert column labels
 - ▣ Improves readability
 - ▣ Print a single header line before looping through data arrays

```
32  
33     f.write('t, y1, y2\n')  
34
```

- ▣ Note the added newline character, `\n`, at the end of the string



```
dataFile1.txt - Notepad  
File Edit Format View Help  
t, y1, y2  
|
```



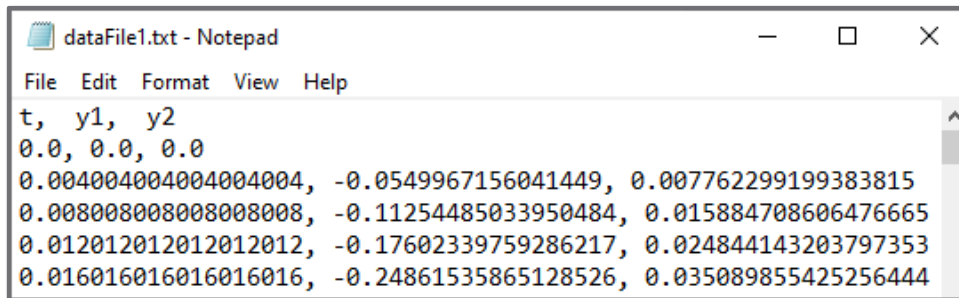
File I/O – Writing – Data

14

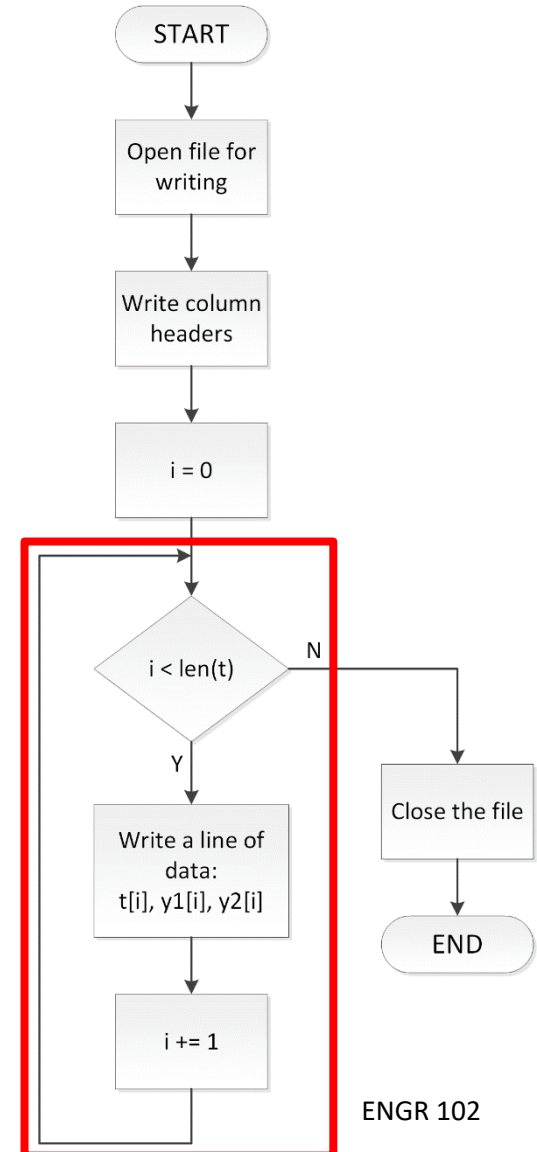
- Write data arrays
 - ▣ Comma-separated columns
 - ▣ Loop through arrays
 - ▣ Print line-by-line

```
34  
35     for i in range(len(t)):  
36         f.write('{} , {} , {}\n'.format(t[i], y1[i], y2[i]))  
37
```

- ▣ Commas, spaces and newlines included in the write string
- ▣ Could control formatting, e.g. precision



```
dataFile1.txt - Notepad  
File Edit Format View Help  
t, y1, y2  
0.0, 0.0, 0.0  
0.004004004004004004, -0.0549967156041449, 0.007762299199383815  
0.008008008008008008, -0.11254485033950484, 0.015884708606476665  
0.012012012012012012, -0.17602339759286217, 0.024844143203797353  
0.016016016016016016, -0.24861535865128526, 0.035089855425256444
```



Closing a text file – `f.close()`

15

- After opening and writing to or reading from a file, that file must be closed

```
f.close()
```

- Access issues may arise if file is not closed

16

Context Managers

File I/O – Context Managers

17

- In the previous example, we saw that we must explicitly close a file when we are done with it
- If a file does not get closed, it may be unavailable to us or other processes later
- Reasons a file would not get closed
 - We forget to close it in our code
 - Computer crashes while we have a file open
- Python provides a better way to access files
 - ***Context managers***

File I/O – Context Managers

18

□ **Context managers:**

- All file I/O code in a block following a ***with*** statement
- File automatically closed when exiting the context manager block

```
with open('dataFile1.txt', 'w') as f:  
    f.write('Hello!')
```

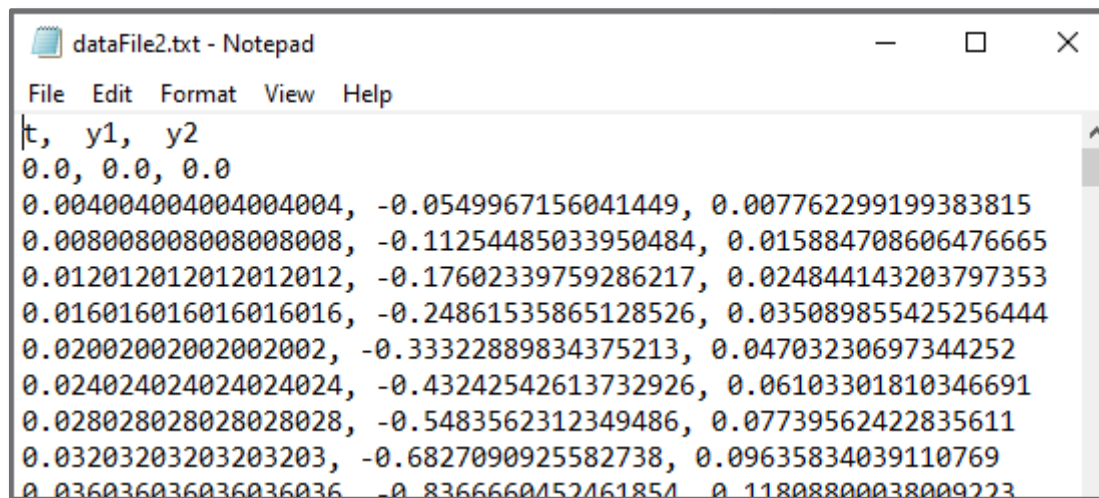
- Preferred file access method

File I/O – Writing – Context Manager

19

- Now, write the same data as before using a context manager:

```
43
44     with open('dataFile2.txt', 'w') as f:
45         f.write('t, y1, y2\n')
46
47         for i in range(len(t)):
48             f.write('{} {}, {}\n'.format(t[i], y1[i], y2[i]))
49
```



```
dataFile2.txt - Notepad
File Edit Format View Help
t, y1, y2
0.0, 0.0, 0.0
0.004004004004004004, -0.0549967156041449, 0.007762299199383815
0.008008008008008008, -0.11254485033950484, 0.015884708606476665
0.012012012012012012, -0.17602339759286217, 0.024844143203797353
0.016016016016016016, -0.24861535865128526, 0.035089855425256444
0.02002002002002002, -0.33322889834375213, 0.04703230697344252
0.024024024024024024, -0.43242542613732926, 0.06103301810346691
0.028028028028028028, -0.5483562312349486, 0.07739562422835611
0.03203203203203203, -0.6827090925582738, 0.09635834039110769
0.036036036036036036, -0.8366660452461854, 0.11808800038009223
```

Exercise – Write to a File

20

Exercise

- Write a script to do the following:
 - ▣ Define an array of angles, x , with 100 values between 0 and 2π
 - ▣ Calculate $y = \sin(x)$
 - ▣ Use a context manager to write x and y to a text file as columns of data
 - ▣ Separate x/y values with a comma and a space
 - ▣ Format values as floating-point numbers with five decimal places
 - ▣ Include column labels: x , $\sin(x)$

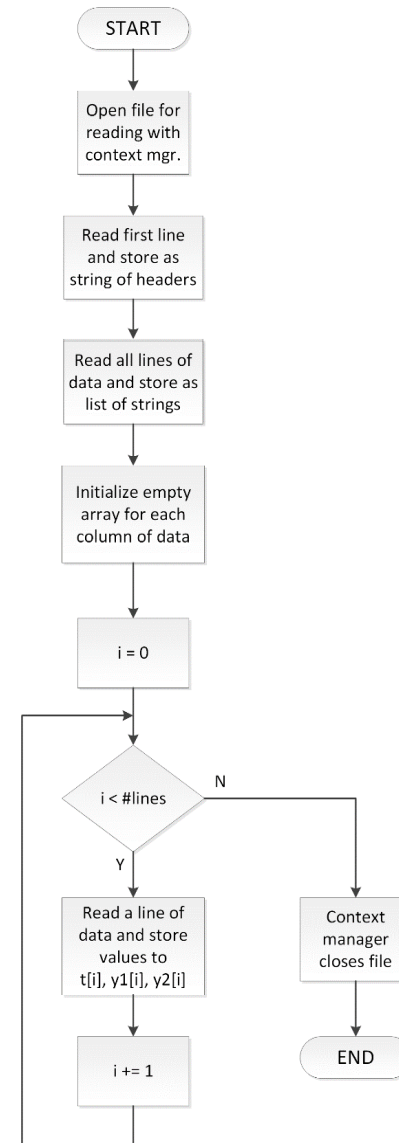
21

Reading from Files

File I/O – Reading

22

- Now, read in the data from the file we just wrote
- The basic procedure:



File I/O – Reading – Headers

23

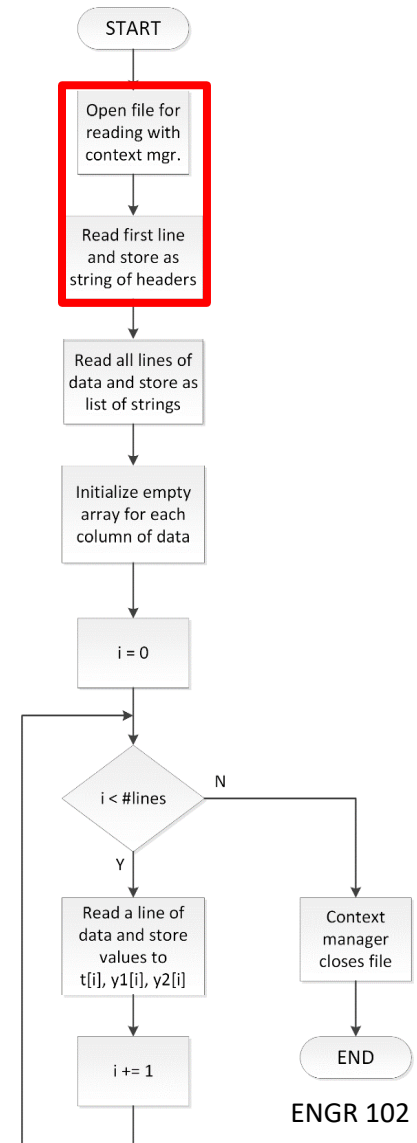
- Read header line(s) first
 - `f.readline()`
 - Whether they'll be used or not
 - Advances reading to next line
 - Stored as a single string
- Strip the `\n` from end of string
 - `.strip()`
- Split into individual strings on commas, if desired
 - `.split(',')`

```
53  
54 with open('dataFile2.txt', 'r') as f:  
55     f_headers = f.readline().strip() # strip() removes \n's  
56
```

```
In [60]: print(f_headers, type(f_headers))  
t, y1, y2 <class 'str'>
```

```
In [61]:
```

Webb



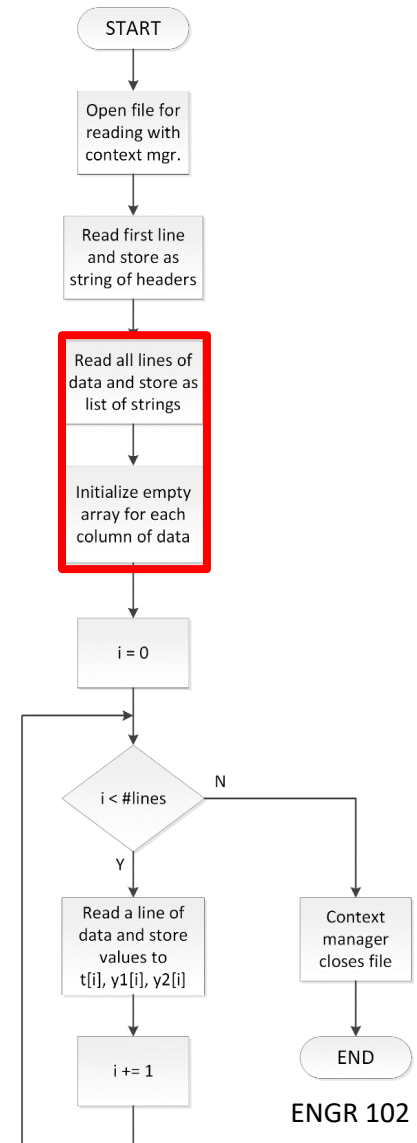
File I/O – Reading – Read Data Lines

24

- Read **all** data lines at once
 - ▣ `f.readlines()`
 - ▣ A list of strings
- Will loop through each string in the list to extract data
- Initialize empty arrays to store data
 - ▣ Same length as list of strings (i.e. number of lines)

```
56
57     # read all lines into list of strings
58     lines = f.readlines()
59
60     # initialize arrays for each variable (each column)
61     t_r = np.empty(len(lines))
62     y1_r = np.empty(len(lines))
63     y2_r = np.empty(len(lines))
64
```

Webb



ENGR 102

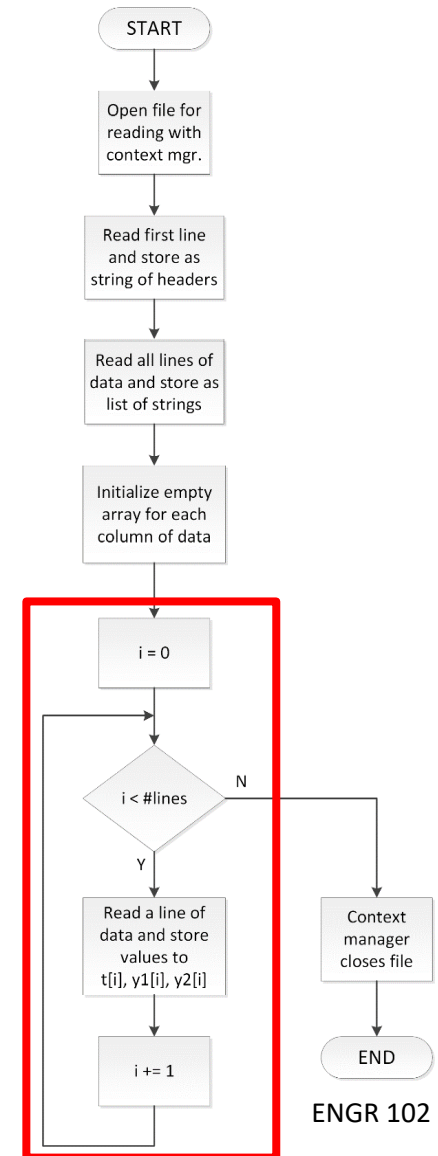
File I/O – Reading – Extract Data

25

- Loop through each string in the list to extract data
 - ▣ Remove `\n` at end of line
 - `.strip()`
 - ▣ Extract string for data value from each column
 - `.split()`
 - ▣ Convert each string to a float
 - ▣ Insert values into data arrays

```
66     for i, line in enumerate(lines):
67         # remove \n's and split on comma/spaces
68         line_strings = line.strip().split(',')
69         # convert individual values to from str to float
70         t_r[i] = float(line_strings[0])
71         y1_r[i] = float(line_strings[1])
72         y2_r[i] = float(line_strings[2])
73
```

Webb

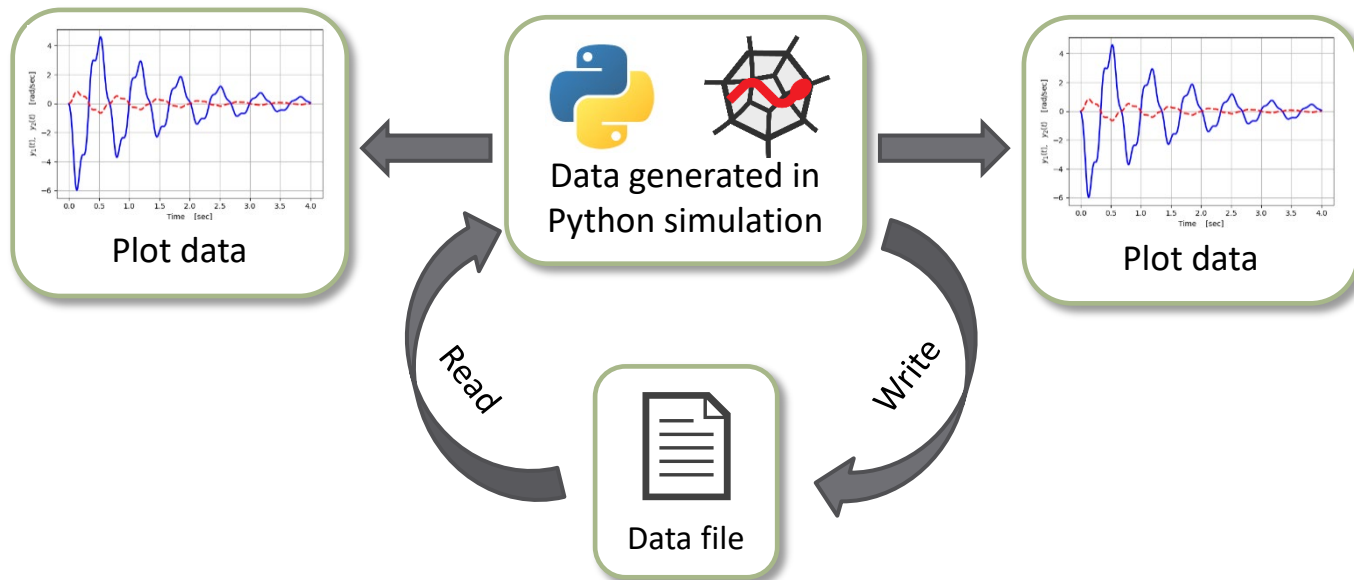


ENGR 102

File I/O - Example

26

- In the previous example, we:
 - ▣ Wrote data generated in Python to a file
 - ▣ Read that same data back into Python
- We can verify that the data we read in is the same as that which we wrote out (e.g. plot it)
- Often only want to read or write, not both – process is the same



27

File I/O Using Pandas

High-Level File I/O Using Pandas

28

- The Pandas package provides many powerful, high-level functions for reading from and writing to files
- We'll introduce functions for reading from/writing to comma-separated-variable (CSV) and Excel files
 - ▣ `to_csv()`
 - ▣ `read_csv()`
 - ▣ `to_excel()`
 - ▣ `read_excel()`
- Typically use these for file I/O
 - ▣ Much simpler than the low-level methods covered previously

Importing the Pandas Package

29

- Just like with other Python packages we have used (e.g. NumPy, Matplotlib), we must ***import*** Pandas before we can use it

```
import pandas as pd
```

```
2  
3 import numpy as np  
4 from matplotlib import pyplot as plt  
5
```

Pandas Data Objects – DataFrame

30

- The Basic Pandas data type is the **DataFrame** object
 - ▣ *A two-dimensional, labeled data structure with columns of possibly different types*

- To write data arrays to a file, first **create a DataFrame**:

- ▣ **Create a dict of the arrays, e.g.:**

```
data = {'t': t, 'y1': y1, 'y2': y2}
```

- ▣ Pass the dict of arrays to `pd.DataFrame()`

```
df1 = pd.DataFrame(data)
```

```
In [70]: data = {'t': t, 'y1': y1, 'y2': y2}
In [71]: df1 = pd.DataFrame(data)
In [72]: df1
Out[72]:
```

	t	y1	y2
0	0.000000	0.000000	0.000000
1	0.004004	-0.054997	0.007762
2	0.008008	-0.112545	0.015885
3	0.012012	-0.176023	0.024844
4	0.016016	-0.248615	0.035090
..
995	3.983984	0.091863	-0.012966
996	3.987988	0.082934	-0.011705
997	3.991992	0.073015	-0.010305
998	3.995996	0.062033	-0.008755
999	4.000000	0.049940	-0.007049

```
[1000 rows x 3 columns]
```

Pandas – `df.to_csv()`

31

- Run the `to_csv()` method on the DataFrame object to write to a CSV file

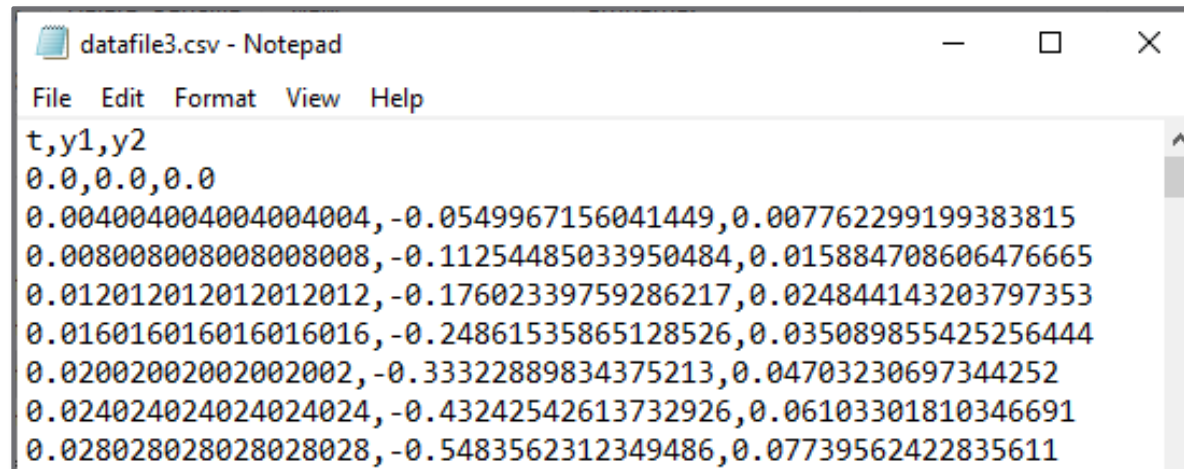
```
df.to_csv(filename, sep, index=False)
```

- `df`: DataFrame object to write
- `filename`: name of file to be written to – a string
- `sep`: field delimiter – optional – default: ','
- `index=False`: prevents writing column of integer indices – optional – default: True

Pandas – df.to_csv()

32

```
30 # %% create a DataFrame object
31 # dict of data arrays
32 data = {'t': t, 'y1': y1, 'y2': y2}
33
34 # creat DataFrame from dict of arrays
35 df1 = pd.DataFrame(data)
36
37
38 # %% write data to a csv file using pandas
39
40 df1.to_csv('datafile3.csv', index=False)
41
```



```
datafile3.csv - Notepad
File Edit Format View Help
t,y1,y2
0.0,0.0,0.0
0.004004004004004004, -0.0549967156041449, 0.007762299199383815
0.008008008008008008, -0.11254485033950484, 0.015884708606476665
0.012012012012012012, -0.17602339759286217, 0.024844143203797353
0.016016016016016016, -0.24861535865128526, 0.035089855425256444
0.02002002002002002, -0.33322889834375213, 0.04703230697344252
0.024024024024024024, -0.43242542613732926, 0.06103301810346691
0.028028028028028028, -0.5483562312349486, 0.07739562422835611
```


Pandas – `pd.read_csv()`

33

- Read from a CSV file

```
df = pd.read_csv(filename)
```

- ▣ `filename`: name of file to be to read from – a string
- ▣ `df`: DataFrame object returned
- ***Index by column labels*** to extract DataFrame data to arrays, e.g.:

```
t = df['t']  
y1 = df['y1']  
y2 = df['y2']
```

Pandas – pd.read_csv()

34

```
43 # %% read from csv file
44
45 df2 = pd.read_csv('dataFile3.csv')
46
47 t_r = df2['t']
48 y1_r = df2['y1']
49 y2_r = df2['y2']
50
```

```
In [101]: df2
Out[101]:
```

	t	y1	y2
0	0.000000	0.000000	0.000000
1	0.004004	-0.054997	0.007762
2	0.008008	-0.112545	0.015885
3	0.012012	-0.176023	0.024844
4	0.016016	-0.248615	0.035090
..
995	3.983984	0.091863	-0.012966
996	3.987988	0.082934	-0.011705
997	3.991992	0.073015	-0.010305
998	3.995996	0.062033	-0.008755
999	4.000000	0.049940	-0.007049

```
[1000 rows x 3 columns]
```

Pandas – `df.to_excel()`

35

- Run the `to_excel()` method on the DataFrame object to write to an Excel file

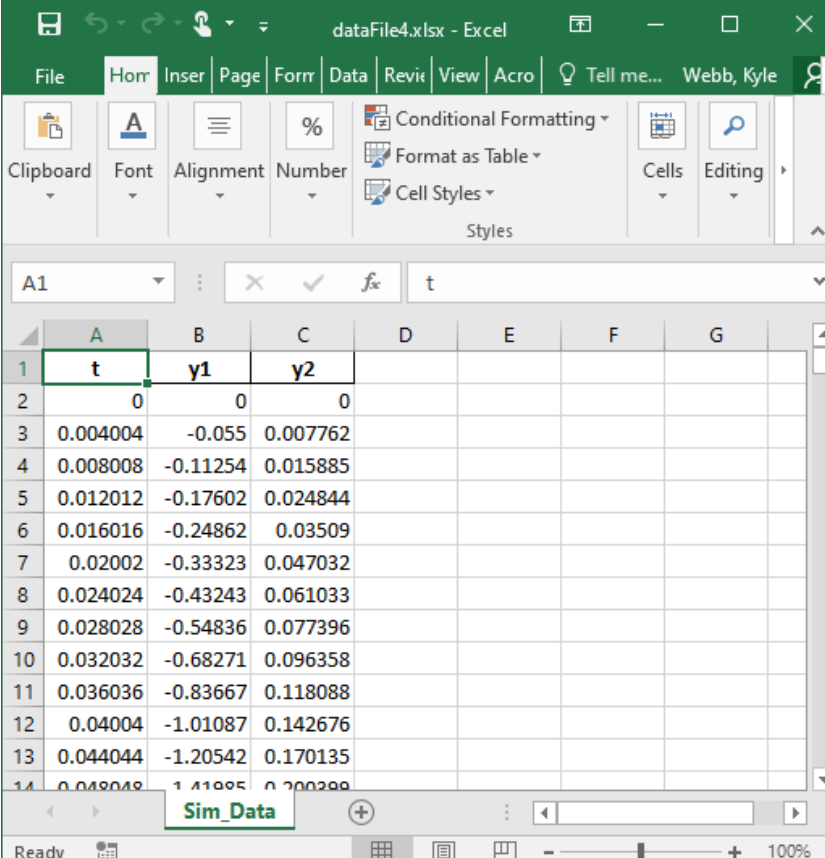
```
df.to_excel(filename, sheet_name='Sheet1', index=False)
```

- `df`: DataFrame object to write
- `filename`: name of file to be written to – a string
- `sheet_name`: name of sheet in the Excel file to be written to – optional – default: 'Sheet1'
- `index=False`: prevents writing column of integer indices – optional – default: True

Pandas – df.to_excel()

36

```
62 # %% write to excel file
63
64 df1.to_excel('dataFile4.xlsx', sheet_name='Sim_Data', index=False)
65
```



The screenshot shows a Microsoft Excel window titled 'dataFile4.xlsx - Excel'. The ribbon includes 'File', 'Home', 'Insert', 'Page Layout', 'Formulas', 'Data', 'Review', 'View', 'Acrobat', 'Tell me...', and 'Webb, Kyle'. The 'Home' ribbon is active, showing 'Clipboard', 'Font', 'Alignment', 'Number', 'Styles', 'Cells', and 'Editing' groups. The active cell is A1, containing the value 't'. The spreadsheet contains the following data:

	A	B	C	D	E	F	G
1	t	y1	y2				
2	0	0	0				
3	0.004004	-0.055	0.007762				
4	0.008008	-0.11254	0.015885				
5	0.012012	-0.17602	0.024844				
6	0.016016	-0.24862	0.03509				
7	0.02002	-0.33323	0.047032				
8	0.024024	-0.43243	0.061033				
9	0.028028	-0.54836	0.077396				
10	0.032032	-0.68271	0.096358				
11	0.036036	-0.83667	0.118088				
12	0.04004	-1.01087	0.142676				
13	0.044044	-1.20542	0.170135				
14	0.048048	-1.41885	0.200288				

Pandas – `df.to_excel()` - `ExcelWriter`

37

- In the previous example a file name was passed to `df.to_excel()`:

```
df.to_excel(filename, sheet_name='Sheet1' ...)
```

- Can specify sheet name, but ***only allowed to write to a single sheet***
- To write to multiple sheets, create an `ExcelWriter` object
 - ▣ Write using context manager:

```
with pd.ExcelWriter(excel_file) as writer:  
    df1.to_excel(writer, sheet_name='Sheet1',  
                index=False)  
    df2.to_excel(writer, sheet_name='Sheet2',  
                index=False)
```

Pandas – df.to_excel()

38

```
67 # %% write to excel file using ExcelWriter object
68 # allows for writing to multiple sheets
69 # create DataFrame objects
70 data_y1 = {'t': t, 'y1': y1}
71 data_y2 = {'t': t, 'y2': y2}
72 df_y1 = pd.DataFrame(data_y1)
73 df_y2 = pd.DataFrame(data_y2)
74
75 with pd.ExcelWriter('dataFile5.xlsx') as writer:
76     df_y1.to_excel(writer, sheet_name='y1(t)', index=False)
77     df_y2.to_excel(writer, sheet_name='y2(t)', index=False)
78
```

	A	B	C	D
1	t	y1		
2	0	0		
3	0.004004	-0.055		
4	0.008008	-0.11254		
5	0.012012	-0.17602		
6	0.016016	-0.24862		
7	0.02002	-0.33323		
8	0.024024	-0.43243		
9	0.028028	-0.54836		
10	0.032032	-0.68271		
11	0.036036	-0.83667		
12	0.04004	-1.01087		
13	0.044044	-1.20542		
14	0.048048	-1.41985		
15	0.052052	-1.65314		
16	0.056056	-1.90376		
17	0.06006	-2.16966		
18	0.064064	-2.44836		
19	0.068068	-2.73699		
20	0.072072	-3.03233		

Pandas – `pd.read_excel()`

39

- Read from an Excel file

```
df = pd.read_excel(filename, sheet_name='Sheet1')
```

- ▣ `filename`: name of file to be to read from – a string
 - ▣ `sheet_name`: name of sheet in the Excel file to read from – optional – default: 'Sheet1'
 - ▣ `df`: DataFrame object returned
- ***Index by column labels*** to extract DataFrame data to arrays

Pandas – pd.read_excel()

40

```
66 # %% read from excel file
67
68 df3 = pd.read_excel('datafile4.xlsx', sheet_name='Sim_Data')
69
70 t_r = df3['t']
71 y1_r = df3['y1']
72 y2_r = df3['y2']
73
```

```
In [102]: df3
Out[102]:
```

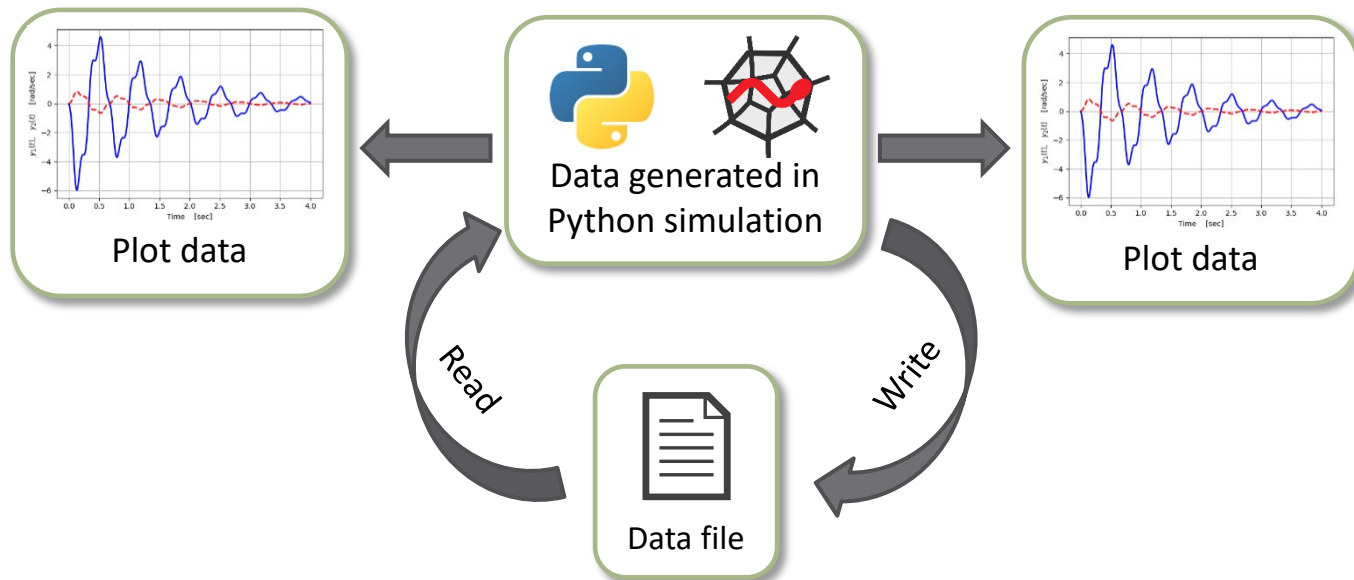
	t	y1	y2
0	0.000000	0.000000	0.000000
1	0.004004	-0.054997	0.007762
2	0.008008	-0.112545	0.015885
3	0.012012	-0.176023	0.024844
4	0.016016	-0.248615	0.035090
..
995	3.983984	0.091863	-0.012966
996	3.987988	0.082934	-0.011705
997	3.991992	0.073015	-0.010305
998	3.995996	0.062033	-0.008755
999	4.000000	0.049940	-0.007049

[1000 rows x 3 columns]

File I/O with Pandas - Example

41

- Again, we've seen how to write to/read from files
 - ▣ Now, using Pandas – much easier
 - ▣ CSV and Excel files
- Just a very brief intro
 - ▣ Pandas is very powerful
 - ▣ Consult the documentation as needed



Exercise – Write Data to Excel

42

Exercise

- Write a script to do the following:
 - ▣ Define an array of angles, x , with 100 values between 0 and 2π
 - ▣ Calculate $y = \sin(x)$
 - ▣ Create a dict containing x and y
 - ▣ Create a DataFrame from the dict
 - ▣ Write the DataFrame to an Excel file
 - Sheet name: $y = \sin(x)$