ENGR 298: Engineering Analysis and Decision Making – Writing Files

Dr. Jason Forsyth
Department of Engineering
James Madison University

Writing Data to Files

- Our analysis can often produce data that should be saved for later processing; either intermediate results to hand off to another process or final analysis to be stored.

- There are many files types (txt, doc, JSON, XML, HDF5…etc) all with different features and properties. Will focus on simply txt/csv files for human readable format.

- Python does support natively writing to JSON. May cover at later date but non-native objects require Serialization.

```
Sample_Name,Material_Type,Tensile_Strength,Fracture_Strain,Elastic_Modulus,Yield_Strength
C01A1045CR_1,1045CR,787.9853962470363,0.1614,204.48360347511277,591.0496055465012
C02A1045CR_1,1045CR,808.2818259006015,0.169,187.2525126541097 2,615.2701167270867
C03A1045CR_1,1045CR,780.832536990291,0.1673,206.07287355644328,581.460615688504
C04A1045CR_1,1045CR,819.5370166505515,0.1584,189.37660166056475,627.5934724063153
C05A1045CR_1,1045CR,799.7138665692523,0.1678,211.22350766615887,596.0294293494273
C06A1045CR_1,1045CR,816.81265666773,0.1682,187.38946860984245,625.1167815128414
C07A1045CR_1,1045CR,785.3230418139093,0.1699,208.5144956704921,605.7125958979988
C01A2024_1,2024,473.9663518834583,0.2641,67.91973865120738,288.4807631553222
C02A2024_1,2024,476.36932476867486,0.2176,64.8677183750668,354.61976103469306
C03A2024_1,2024,466.5187365129491,0.192,72.6194161172565,354.40768936140785
C04A2024_1,2024,458.84388068219886,0.1936,69.34127686907144,352.20943726267245
C05A2024_1,2024,463.8319793916702,0.2481,69.63816298215642,354.40768936140785
C06A2024_1,2024,464.6931363115216,0.2305,109.65850256999525,350.7303375543619
C07A2024_1,2024,446.4674305430813,0.2379,68.19989930707435,344.10884909451295
C01APMMA_1,PMMA,81.36236958062278,0.05057,3.3345726004633147,47.60710492486 0906
C02APMMA_1,PMMA,80.46769490450241,0.0512,3.104164236093 0304,52.08096308073987
C03APMMA_1,PMMA,79.20086657554037,0.06983,2.7401631572014935,45.48952546628137
C04APMMA_1,PMMA,79.0018504894893,0.04545,3.2216074851617704,51.933645498867996
C05APMMA_1,PMMA,77.01033808438562,0.05568,3.1346379779181106,46.47209282218446
C06APMMA_1,PMMA,79.33943345032885,0.05302,2.933374650143233,53.209679918783436
C07APMMA_1,PMMA,75.5981278310394,0.05304,3.179665313074314,49.0922642107 8233
```

Writing to Files…

- To access any file it must first be opened via the open() method.

- If file path passed to method does not exist, then program will throw a file not found error. If you encounter this, you are trying to open a file that does not exist. Check the file path.

- Often helpful to check if an file exists before attempting to open()

```
>>> open('random_data.txt')
Traceback (most recent call last):
  File "<input>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'random_data.txt'
```

```python
# to open a file, first we need a file name
file_name = "my_data.txt"

# If you attempt to open a file and that file does not exists
# an error will be returned. It is always good to check if the file
# actually exists before opening it. We will use os.path() for this

print("Does this file exist: ", file_name)
if os.path.exists(file_name) == False:
    print("File does not exist! Error!")
else:
    print("It does exist!")
```

This solution checks whether a file exists before an accidental exception is thrown for a non-existent file

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

Open *file* and return a corresponding file object. If the file cannot be opened, an `OSError` is raised. See Reading and Writing Files for more examples of how to use this function.

file is a path-like object giving the pathname (absolute or relative to the current working directory) of the file to be opened or an integer file descriptor of the file to be wrapped. (If a file descriptor is given, it is closed when the returned I/O object is closed unless *closefd* is set to `False`.)

mode is an optional string that specifies the mode in which the file is opened. It defaults to `'r'` which means open for reading in text mode. Other common values are `'w'` for writing (truncating the file if it already exists), `'x'` for exclusive creation, and `'a'` for appending (which on *some* Unix systems, means that *all* writes append to the end of the file regardless of the current seek position). In text mode, if *encoding* is not specified the encoding used is platform-dependent: `locale.getpreferredencoding(False)` is called to get the current locale encoding. (For reading and writing raw bytes use binary mode and leave *encoding* unspecified.) The available modes are:

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open for updating (reading and writing)

The default mode is `'r'` (open for reading text, a synonym of `'rt'`). Modes `'w+'` and `'w+b'` open and truncate the file. Modes `'r+'` and `'r+b'` open the file with no truncation.

General File I/O Guidelines

- Typical process will open() a file, read/write data, and then close(). Can pass parameter to open() to indicate read/write/append…etc.

- A file that is open cannot be read by other processes in the computer (generally). Holding a file ‘handle’ can prevent access.

- Closing the file releases the handle. Also ensures all data that was written has been *flushed* and is no longer in a *buffer*. Close() commits all changes to the file.

- Should not access an open file while being written. Unsure of results.

```python
# Typically files are accessed with open(). If our file is new, we can create
# it as we open the file by passing the 'w' parameter to open()
my_new_file = open(file_name, 'w')

# now that we have created the file. This code should not error.
print("Attempting to open file: ", file_name)
if os.path.exists(file_name) == False:
    print("File does not exist! Error!")
else:
    print("It does exist!")

# Having opened the file, we now have a 'handle' object through which we can read/write/modify the file
# we will leave this file alone for now. So let's close it.
my_new_file.close()
```

Write textual data to a file

- Current examples only consider ‘string-like’ data: letters, numbers, white space characters that are written to a human-readable file.

- **write()**: takes a string and writes to file (easy enough…)

- **writelines()**: will write multiple strings in single call. Don’t use if unsure of results.

```
# use the write method from the handle to write to the file
string_to_be_written = "Hello world!"

# given a string, write it out to the file
hello_world_file.write(string_to_be_written)
```

Constructing Strings and Line Endings

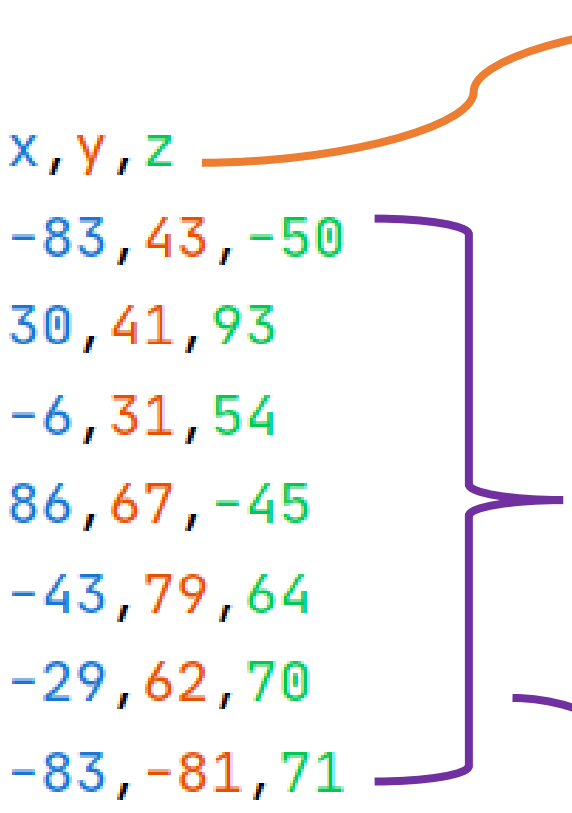
- Since data to be written must be a ‘string’ all objects that go out to the file must be converted to str().
- Easy enough for basic numbers: str(7) or str (3.14159)
- Challenge is to ensure data that is written out, can be easily read back in by another/future program. File structure should be rational.
- Also, line endings ‘\n’ are required to place each string on a new line.

Strings must/should be constructed before being written to double check format

```
# stitch those integers into a string to write
# each integer must be manually converted to a string and
# each element must be separated by a comma and end with a new line '\n'
string_to_write = str(x)+","+str(y)+","+str(z)+'\n'  string_to_write: '-83,43,-50\n'

# write the CSV line to the file
my_csv.write(string_to_write)
```

```
x,y,z
-83,43,-50
30,41,93
-6,31,54
86,67,-45
-43,79,64
-29,62,70
-83,-81,71
```




```python
# construct a silly string as headers for printing out a vector in 3-space <x,y,z>
header_string = "x,y,z\n"

# write header to file
my_csv.write(header_string)

# fill the remainder of the file with random x,y,z values for several lines
num_lines_to_write = random.randint(5,10)

# for each line, generate random ints and write to file
for idx in range(0,num_lines_to_write):

    # generate three random integers
    x=random.randint(-100,100)
    y=random.randint(-100,100)
    z=random.randint(-100,100)

    # stitch those integers into a string to write
    # each integer must be manually converted to a string and
    # each element must be separated by a comma and end with a new line '\n'
    string_to_write = str(x)+","+str(y)+","+str(z)+'\n'

    # write the CSV line to the file
    my_csv.write(string_to_write)

# after we're done, close the file
my_csv.close()
```

Most things can be turned into strings but they may not be a regular format…

```
>>> my_list=[3,4,5,6]
>>> str(my_list)
'[3, 4, 5, 6]'
```

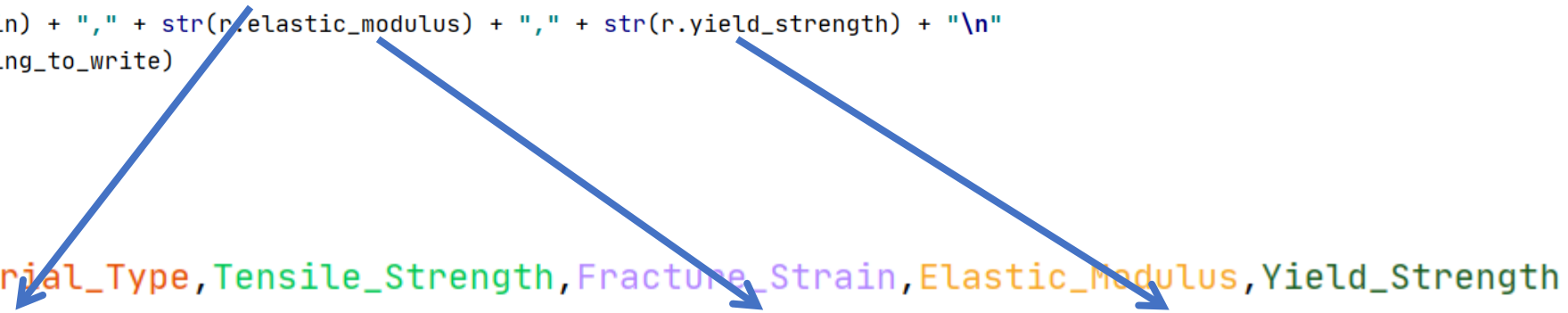
```
>>> matrix = np.random.rand(3,3)
>>> str(matrix)
'[[0.02095137 0.17021161 0.26403264]\n [0.4627143  0.93072922 0.22628211]\n [0.12350559 0.17859182 0.7443263 ]]'
```

These strings could be written to a file, but wouldn't you rather have a CSV file than these odd formats? Numpy or Pandas won't automatically parse these.

Writing to files is easy, but the main challenge is formatting the output correctly…

```
file.writelines("Sample_Name,Material_Type,Tensile_Strength,Fracture_Strain,Elastic_Modulus,Yield_Strength\n")
for r in results:
    string_to_write = r.name + "," + r.material_type + "," + str(r.tensile_strength) + "," + str(
        r.fracture_strain) + "," + str(r.elastic_modulus) + "," + str(r.yield_strength) + "\n"
    file.writelines(string_to_write)

file.close()
```



```
Sample_Name,Material_Type,Tensile_Strength,Fracture_Strain,Elastic_Modulus,Yield_Strength
C01A1045CR_1,1045CR,787.9853962470363,0.1614,204.48360347511277,591.0496055465012
C02A1045CR_1,1045CR,808.2818259006015,0.169,187.25251265410972,615.2701167270867
C03A1045CR_1,1045CR,780.832536990291,0.1673,206.07287355644328,581.460615688504
C04A1045CR_1,1045CR,819.5370166505515,0.1584,189.37660166056475,627.5934724063153
C05A1045CR_1,1045CR,799.7138665692523,0.1678,211.22350766615887,596.0294293494273
C06A1045CR_1,1045CR,816.81265666773,0.1682,187.38946860984245,625.1167815128414
C07A1045CR_1,1045CR,785.3230418139093,0.1699,208.5144956704921,605.7125958979988
C01A2024_1,2024,473.9663518834583,0.2641,67.91973865120738,288.4807631553222
```

Summary

- Writing to files is easy but more challenging is structuring the data so it can easily be viewed by human or read back by program.

- Take care with formatting strings and line endings; don’t hold files open longer than necessary for reading/writing.

- Gradescope assignment will generate CSV from tensile data. Will be forced to use in own program for later analysis.