Berkeley
UNIVERSITY OF CALIFORNIA

M. Hohle:

Physics 77: Introduction to Computational Techniques in Physics

syllabus:

- Introduction to Unix & Python	(week 1 - 2)
- Functions, Loops, Lists and Arrays	(week 3 - 4)
- Visualization	(week 5)
- Parsing, Data Processing and File I/O	**(week 6)**
- Statistics and Probability, Interpreting Measurements	(week 7 - 8)
- Random Numbers, Simulation	(week 9)
- Numerical Integration and Differentiation	(week 10)
- Root Finding, Interpolation	(week 11)
- Systems of Linear Equations	(week 12)
- Ordinary Differential Equations	(week 13)
- Fourier Transformation and Signal Processing	(week 14)
- Capstone Project Presentations	(week 15)

<u>most common types:</u>

plain text: `.dat, .txt, .fa`

`.csv`

`.xls`

`.xlsx`

python: `.py, .npy`

```
import pandas as pd
```

```
pd.re
    read_clipboard
    read_csv
    read_excel
    read_feather
    read_fwf
    read_gbq
    read_hdf
```

Excel and CSV

Let us read the two data sets (same content): *'Data_set_0.xlsx' (127 MB)*
'Data_set_0.csv' (182 MB)

We also want to benchmark different parsing methods concerning **runtime** and efficiency!

```
import pandas as pd
from my_timer import my_timer

@my_timer
def PandasExcel():
        df = pd.read_excel('Data_set_0.xlsx')
        return df
```

```
Console 1/A
In [15]: runfile('C:/Users/MMH_
ParseFiles.py', wdir='C:/Users/
Reloaded modules: my_timer

In [16]: df = PandasExcel()
```

Total runtime: 104 seconds

pandas excel 104sec returns `data frame`

important commands:

`df.values`	extracts only values as `np.array`
`df.corr()`	calculates correlation coefficients
`df.index`	returns rows
`df.columns`	returns columns
`df.head`	shows header
`df[['t1', 't4']]`	returns **data frame** of **selected columns**
`df.loc[['Data Set 12', 'Data Set 2']]`	returns **data frame** of **selected rows**
`df.iloc[4:6, 5:9]`	slicing **data frame** using `iloc`
`df.iloc[4,5] = 999`	manipulating individual entries
`df.insert(2, 'Test', df.iloc[:,1])`	inserting another column called test
`df.rename(index = {1: 'bbb'})`	changing row name
`df.rename(columns = {'time': 'AAA'})`	changing column name
`df.to_excel('Test.xlsx')`	saving df as excel file

pandas excel 104sec

```
@my_timer
def PandasCSV():
    df = pd.read_csv('Data_set_0.csv')
    return df
```

```
In [19]: df = PandasCSV()
Total runtime: 1.3910000000032596 seconds
```

1.4 seconds! That is a factor of 100!!

pandas	**excel**	**104.0sec**
pandas	**csv**	**1.4sec**

dask is a faster alternative to pandas!

```
In [22]: pip install dask
```

```
import dask.dataframe as dd

@my_timer
def DaskCSV():
    df = dd.read_csv('Data_set_0.csv')
    return df
```

```
In [23]: df = DaskCSV()
Total runtime: 0.01600000000325963 seconds
```

```
df = pd.DataFrame(df)
```

Again, we gain a factor of 100!!

However, we might need to transform the output

pandas	**excel**	**104.00sec**
pandas	**csv**	**1.39sec**
dask	**csv**	**0.02sec**

As of July 2024, dask **doesn't have an excel API**

→ alternative **polars**:

```
pip install polars
pip install xlsx2csv  #for excel API
pip install fastexcel
```

```
import polars as pl
```

pandas	**excel**	**104.00sec**
pandas	**csv**	**1.39sec**
dask	**csv**	**0.02sec**

```
@my_timer
def PolarsExcel():
    df = pl.read_excel('Data_set_0.xlsx')
    return df


@my_timer
def PolarsCSV():
    df = pl.read_csv('Data_set_0.csv')
    return df
```

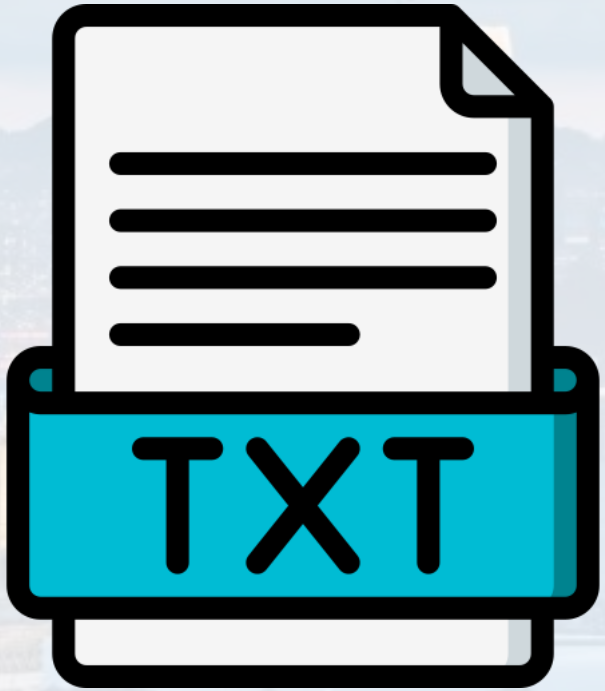
```
In [2]: df = PolarsExcel()
Total runtime: 10.171999999962281 seconds

In [3]: df = PolarsCSV()
Total runtime: 0.14100000000325963 seconds
```

pandas	**excel**	**104.00sec**
polars	**excel**	**10.17sec**
pandas	**csv**	**1.39sec**
polars	**csv**	**0.14sec**
dask	**csv**	**0.02sec**

factor of 4000 – 5000!!

Text Files



Pandas can also read **text files**:

```
raw = pd.read_csv('raw.txt')
```

raw - DataFrame

Index	No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
0	1	2010	1	1	0	nan	-21	-11	1021	NW	1.79	0	0
1	2	2010	1	1	1	nan	-21	-12	1020	NW	4.92	0	0
2	3	2010	1	1	2	nan	-21	-11	1019	NW	6.71	0	0
3	4	2010	1	1	3	nan	-21	-14	1019	NW	9.84	0	0

```
cf = pd.read_csv('cystfibr.txt')
```

cf - DataFrame

Index	age sex height weight bmp fev1 rv frc tlc pemax
0	7 0 109 13.1 68 32 258 183 137 95
1	7 1 112 12.9 65 19 449 245 134 85
2	8 0 124 14.1 64 22 441 268 147 100

Pandas can also read **text files**:

```
cf = pd.read_csv('cystfibr.txt')
```

cf - DataFrame

Index	age sex height weight bmp fev1 rv frc tlc pemax
0	7 0 109 13.1 68 32 258 183 137 95
1	7 1 112 12.9 65 19 449 245 134 85
2	8 0 124 14.1 64 22 441 268 147 100

```
cf = pd.read_csv('cystfibr.txt', delim_whitespace = True)
```

cf - DataFrame

Index	age	sex	height	weight	bmp	fev1	rv	frc	tlc	pemax
0	7	0	109	13.1	68	32	258	183	137	95
1	7	1	112	12.9	65	19	449	245	134	85
2	8	0	124	14.1	64	22	441	268	147	100
3	8	1	125	16.2	67	41	234	146	124	85

Pandas can also read **text files**:

```
cf = pd.read_csv('cystfibr.txt')
```

```
df = PandasCSV()
df.to_csv('Data_set_0.txt')
```

We can save a `.csv` file as `.txt`

```
@my_timer
def PandasTxt():
    df = pd.read_csv('Data_set_0.txt')
    return df
```

```
In [54]: df = PandasTxt()
Total runtime: 1.3439999999827705 seconds
```

as fast as reading a csv

```python
@my_timer
def PandasTxt():
    df = pd.read_csv('Data_set_0.txt')
    return df
```

```
In [54]: df = PandasTxt()
Total runtime: 1.3439999999827705 seconds
```

```python
@my_timer
def DaskTxt():
    df = dd.read_csv('Data_set_0.txt')
    return df
```

```
In [58]: df = DaskTxt()
Total runtime: 0.01600000000325963 seconds
```

```python
df = pd.DataFrame(df)
```

However, we might need to transform the output

sometimes txt files don't come in a nice format

List.txt

File Edit View

```
2024-2-Physics-77-4.pptx
cystfibr.txt
Data_set_0.csv
Data_set_0.txt
Data_set_0.xlsx
List.txt
my_timer.py
ParseFiles.py
raw.txt
```

→ using open

```
with open('List.txt', errors = "ignore") as f:
        text = f.read()
```

reads file line by line and stops automatically when have reached the end

Text editor - text

```
2024-2-Physics-77-4.pptx
cystfibr.txt
Data_set_0.csv
Data_set_0.txt
Data_set_0.xlsx
List.txt
my_timer.py
ParseFiles.py
raw.txt
```

sometimes txt files don't come in a nice format

List.txt

File Edit View

```
2024-2-Physics-77-4.pptx
cystfibr.txt
Data_set_0.csv
Data_set_0.txt
Data_set_0.xlsx
List.txt
my_timer.py
ParseFiles.py
raw.txt
```

```
with open('List.txt', errors = "ignore") as f:
        text = f.read()
```

Text editor - text

```
2024-2-Physics-77-4.pptx
cystfibr.txt
Data_set_0.csv
Data_set_0.txt
Data_set_0.xlsx
List.txt
my_timer.py
ParseFiles.py
raw.txt
```

```
T = list(text.split())
```

Index	Type	Size	
0	str	24	2024-2-Physics-77-4.pptx
1	str	12	cystfibr.txt
2	str	14	Data_set_0.csv
3	str	14	Data_set_0.txt
4	str	15	Data_set_0.xlsx
5	str	8	List.txt
6	str	11	my_timer.py

syntax of open

```
with open('my_file.txt', flags, errors = "ignore") as f:
        text = f.read()
```

finishes once it has reached the end and closes *f*

flags/modes:

'r'	opens file for reading only.
'w'	opens file for writing. If the file exists, it overwrites it, otherwise, it creates a new file.
'a'	opens file for appending only. If the file doesn't exist, it creates the file.
'x'	creates new file. If the file exists, it fails.
'+'	opens file for updating
't'	opens file in text mode (default)
'b'	opens file in binary mode

flags can be combined like, eg 'wb'

`with` opens a loop for reading the file line by line

syntax of open

```
with open('List.txt', 'r') as read_f:
    with open('List_copy.txt', 'w') as write_f:
        for r in read_f:
            write_f.write(r)
```

opens file line by line

opens **new** file line by line for **writing**

writes each line to **new** file

```
In [5]: print(r)
raw.txt
```

```
In [6]: print(type(r))
<class 'str'>
```

.npy Files

```
R1 = np.random.normal(0, 1, (10, 50))

np.save('my_file.npy', R1)
```

my_file.npy

```
R2 = np.load('my_file.npy')
```

R1	Array of float64	(10, 50)
R2	Array of float64	(10, 50)

Thank you for your attention!