Crash Course in Python

Øystein Bjørndal

March 25, 2020

Hello world, complete how-to

1. Download and install python
   `https://www.anaconda.com/download/`
   Anaconda contains python and a bunch of useful packages.
   At TI, consider installing from
   `http://software.itg.ti.com/` and fixing proxy:
   `https://infolink.sc.ti.com/business_rooms/characterization_corner/f/2782/t/80190` (first hit when searcing for "proxy" at `myinfolink.ti.com/`
2. Add python to Path, either during the install or manually in Windows "Environment variables for your account"

Hello world, complete how-to

3. Write a file "hello-world.py" with

```
print("hello world")
```

4.
   - Either use Spyder (similar to the Matlab interface with a interactive session and editor)
   - use Command prompt, type "python hello-world.py" (assuming python is on your Path and you "cd" to the correct folder).

5. "hello world" !!

Some python basics

Demo:
https://github.com/ehmatthes/pcc/releases/download/v1.0.0/beginners_python_cheat_sheet_pcc_all.pdf

Data science libraries

Some great libraries:

- Matplotlib - similar plotting interface to Matlab, just better
- Numpy - similar optimized array/matrix operations as Matlab
- Pandas - Uses the above 2 libraries and much more, useful for data analysis, file reading/writing

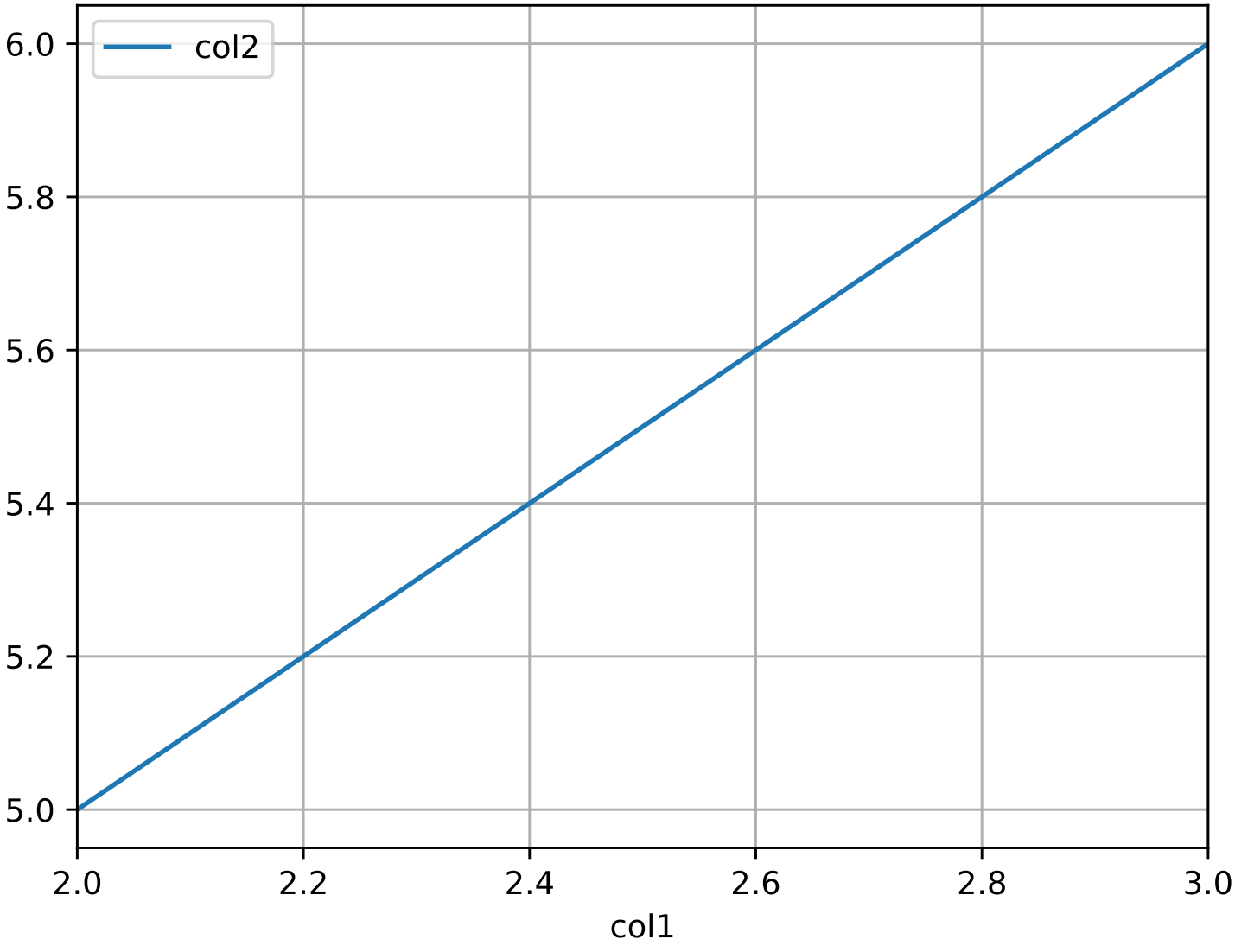
Pandas documentation

Short intro to Pandas
https://pandas.pydata.org/pandas-docs/stable/10min.html

API
https://pandas.pydata.org/pandas-docs/stable/api.html

Pandas

```
import pandas as pd
df = pd.DataFrame([[2, 5, 10], [3, 6, 11]],
                  columns=('col1', 'col2', 'col3'))
df['col1'] #  [2, 3]
df.plot('col1', 'col2', grid=True)
```

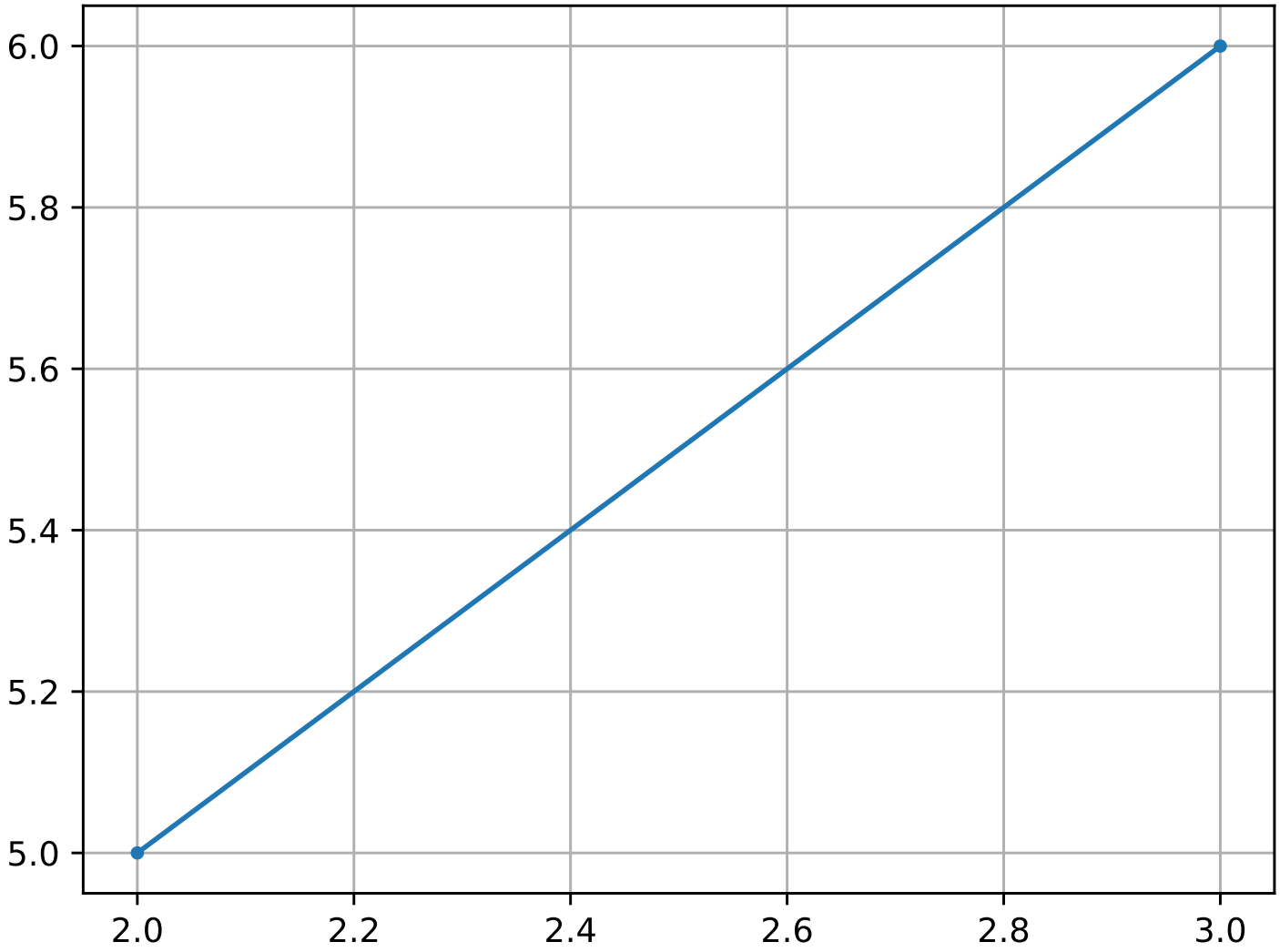

col2
6.0
5.8
5.6
5.4
5.2
5.0
2.0
2.2
2.4
2.6
2.8
3.0
col1

Plotting with Pylab

```
import pylab as plt
import pandas as pd

df = pd.DataFrame([[2, 5, 10], [3, 6, 11]],
                  columns=('col1', 'col2', 'col3'))
df['col1'] #  [2, 3]

plt.figure(2)
plt.plot(df['col1'].values, df['col2'].values,
     marker='.')
plt.grid(True)
```


6.0
5.8
5.6
5.4
5.2
5.0
2.0
2.2
2.4
2.6
2.8
3.0

Pandas indexing

```
>>> import pandas as pd

>>> df = pd.DataFrame([[2, 5, 10], [3, 6, 11]],
...                   columns=('col1', 'col2', 'col3'))
>>> df['col1']
0    2
1    3
Name: col1, dtype: int64
>>> df['col1'].iloc[0]
2
>>> df['col1'].iloc[1]
3
>>> df['col1'].iloc[-1] # last element
3
```

Pandas indexing slize

```
>>> import pandas as pd

>>> df = pd.DataFrame([[2, 5, 10], [3, 6, 11]],
...                   columns=('col1', 'col2', 'col3'))
>>> df
   col1  col2  col3
0     2     5    10
1     3     6    11
>>> # start at index 0, end at index 2 (not inclusive)
>>> df['col1'].iloc[0:2]
0    2
1    3
Name: col1, dtype: int64
```

Pandas return submatch

```
>>> df = pd.DataFrame([[2, 5, 10], [3, 6, 11], [4, 5, 1
...                    columns=('col1', 'col2', 'col3'))
>>> df
   col1  col2  col3
0     2     5    10
1     3     6    11
2     4     5    12
>>> df['col2'] == 5
0     True
1    False
2     True
Name: col2, dtype: bool
>>> df[df['col2'] == 5]
   col1  col2  col3
0     2     5    10
2     4     5    12
```

Pandas reading and writing to excel

```
>>> root = r'R:\CC2652R\PG2.1_Q1\Debug\OB_OPT2\TX_BLE_C
>>> root_tmp = r'c:\tmp'
>>> filename = 'TX_BLE_quality_frequencySweep_TC029_TxP
>>> full_path_in = os.path.join(root, filename)
>>> full_path_out = os.path.join(root_tmp, filename)

>>>  # read
>>> df = pd.read_excel(full_path_in)
>>> # write
>>> df.to_excel(full_path_out, index=False)
>>> # see
>>> # svn\lab\Python\py_excel_utilities\pandas_util.py
>>> # for more formats
```

Getting filenames

```
>>> # single filename
>>> full_path = r'c:\tmp\myfile.xlsx'
>>> # OR
>>> root = r'c:\tmp'
>>> filename = 'myfile.xlsx'
>>> full_path = os.path.join(root, filename)
```

Getting files continued

```
>>> # multiple filenames, wildcard
>>> from glob import glob
>>> for full_path in glob(r'c:\tmp\*.xlsx'):
...     print(full_path)
...     break;# break loop
...
c:\tmp\concatenated-ToF_corr_comp_vs_CableLength.xlsx
>>> # multiple filenames, all subfolders
>>> for root, dirs, files in os.walk(r'c:\tmp'):
...     for filename in files:
...         if filename.endswith('.xlsx'):
...             full_path = os.path.join(root, filename)
...
```

DataFrame Grouping

```
>>> df['EM name'].unique()
array(['E0_1019', 'E0_1020', 'E0_1021', 'E0_1022', 'E0_
       dtype=object)
>>> for temperature, group in df.groupby('Temperature'):
...     for em_name, group in group.groupby('EM name'):
...         print(temperature, em_name)
...
25 E0_1019
25 E0_1020
25 E0_1021
25 E0_1022
25 E0_1023
25 E0_1024
>>> group['EM name'].unique()
array(['E0_1024'], dtype=object)
```

Demo

`https://dosfiler01.itg.ti.com/svn/lab/Python/PythonCrashCourse/cleanup_tof_results.py`

PEP 8

Write better code
`https://www.python.org/dev/peps/pep-0008/?`
Not intended for beginner python users, but shows some interesting syntax and use cases and describes best-practices.

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to d
Although that way may not be obvious at first unless you're Dutc
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```