Crash Course in Python

Øystein Bjørndal

July 18, 2018

Python 2 vs 3

Python 2

Python 3

```python
print "hello world"
```

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

```
3/2 == 1
```

```
3/2 == 1.5
```

Lazy evaluation:

```python
>>> map(str.upper, ['foo', 'baz'])
['FOO', 'BAZ']
```

```python
>>> map(str.upper, ['foo', 'ba
<map at 0x2a7c10647f0>
```

USE PYTHON 3!

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
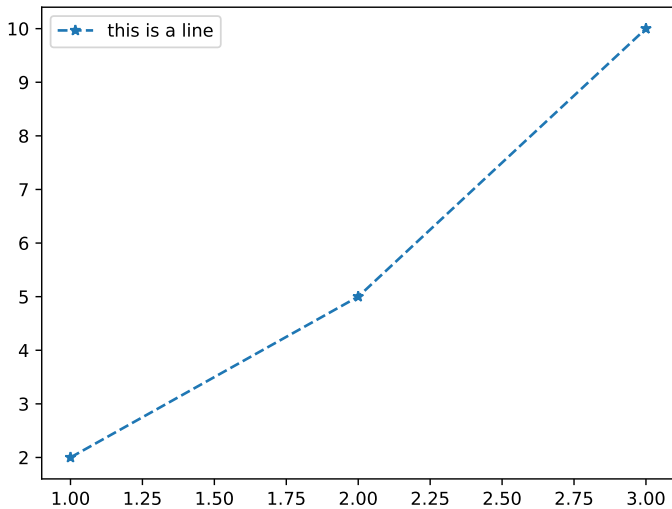
Data science libraries - Matplotlib

```python
import pylab as plt
# Alternative: import pylab
# Please don't: from pylab import *

x = [1, 2, 3]
y = [2, 5, 10]

plt.plot(x, y, linestyle='--', marker='*',
    label='this is a line')
plt.legend(loc='best')
```

matplotlib documentation

`https://matplotlib.org/api/pyplot_summary.html`

Data science libraries - Numpy

In the previous example, x and y, where "native" python lists, these do not work as expected from a Matlab viewpoint:

```
>>> x = [1, 2, 3]
>>> y = [2, 5, 10]
>>> x + y
[1, 2, 3, 2, 5, 10]
>>> x*5
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> # workaround
>>> for ix in range(len(x)):
...     x[ix] *= 5
...
>>> x
[5, 10, 15]
```

Numpy to the rescue

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([2, 5, 10])
>>> x + y
array([ 3,  7, 13])
>>> x*5
array([ 5, 10, 15])
>>> # Still works the same:
>>> for ix in range(len(x)):
...     x[ix] *= 5
...
>>> x
array([ 5, 10, 15])
```

Numpy documentation

cheat sheet
`https://www.datacamp.com/community/blog/`
`python-numpy-cheat-sheet`

Numpy for Matlab users
`https://docs.scipy.org/doc/numpy-1.14.0/user/`
`numpy-for-matlab-users.html`

Documentation (or just google "numpy x")
`https://docs.scipy.org/doc/numpy-1.14.0/`
`reference/index.html`

Pandas documentation

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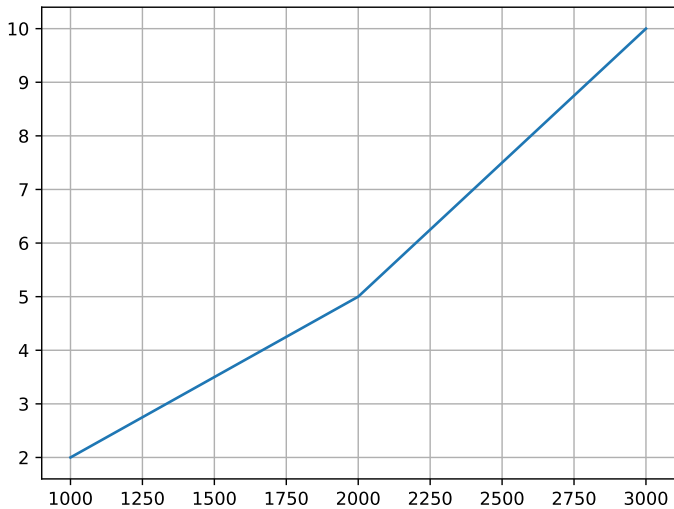
API
```
https:
//pandas.pydata.org/pandas-docs/stable/api.html
```

Pandas

```python
import pandas as pd
s = pd.Series([2, 5, 10], index=(1000,2000,3000))
s[1000] # 2
s.plot(grid=True)
```

Pandas indexing series

```
>>> import pandas as pd
>>> s = pd.Series([2, 5, 10], index=(1000,2000,3000))
>>> s[1000]
2
>>> s.iloc[0]
2
>>> s[3000]
10
>>> s.iloc[-1]
10
```

Pandas indexing series, using strings

```python
>>> import pandas as pd
>>> s = pd.Series([2, 5, 10], index=('Experiment 1',
...                         'Debug',
...                         'Debug-Debug'))
>>> s['Experiment 1']
2
>>> s.iloc[0]
2
>>> s['Debug-Debug']
10
>>> s.iloc[-1]
10
```

Pandas as spotfire replacement Demo

Internal TI link: `https:`
`//oslosvn.norway.design.ti.com/svn/lab/Python/`
`PythonCrashCourse/pandas_as_spotfire_demo.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!
```