Python for data science

Ragnhild C. Noven

Basic imports

```
import numpy as np
import scipy as sp
```

Navigation

```
import os
os.getcwd()
```

Input and output

- **Read** lines from file

```
import sys
with open(sys.argv[1]) as f:
    flines = f.readlines()
for line in flines:
    print line
```

- **Write** to file

```
with open('filename.txt','w') as f:
    f.write(mystr)
```

Lists

- **Merge** a list into another using `mylist.extend(taillist)` to

Dictionaries

- Iterate over key,value pairs in dictionary

```
[func(key,value) for key,value in dict.iteritems
    ()]
[func(key,value) for key,value in zip(dict.keys
    (),dict.values())]
```

- Get the value at key, and if key is not in dictionary, insert it with a default value:

```
mydict.setdefault(key,defval)
```

Strings

- Basic functions:

```
>>> "%s is %.2f" % ("Foo",3.567)
'Foo is 3.57'
str.count(substr)
separator.join(iterable)
>>> 'Py' in 'Python'
True
# returns -1 if not found
str.find(substr)
str.replace(old,new)
str.split(sepstr)
str.strip(schars)

str.startswith("substr")
```

Unicode

- Unicode looks like `\u2119`, has integers that uniquely represent symbols.
- The conversion from unicode to bytes can be done by different *encodings*, such as 'ascii', 'utf8', 'windows-1252' etc.
- `unicode.encode -> bytes`
- `bytes.decode -> unicode`
- If you have a unicode string and get errors, try using `'xmlcharrefreplace'` as the second argument to `encode`.

See `https://nedbatchelder.com/text/unipain.html`

Objects

- **Introspection**: use `dir()` or `type()`.

Don't ask permission

```
try:
    do_something(x)
except Exception, e:
    print e
```

Operators

Operator	Meaning
	and, or, not, in
**	Exponent
%	Modulus
!=	Not equal
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	Bitwise XOR

Graphs

Basic graphs

Start with importing the pyplot package

```
import matplotlib.pyplot as plt
```

Then you can use

```
##create empty plot
plt.subplot(1,1,1)
plt.plot([1,2,3])
plt.show()
##NB: clear plot
plt.close()
```

to plot lists.

Barplot

```
plot = plt.subplot()
b2 = plot.bar([1,2,3],[4,5,6],color='g',width
    =0.5)
```

```
b1 = plot.bar([0.5,1.5,2.5],[1,2,3],width=0.5)
plt.show()
```

Histogram

Let `mydict` be a dictionary of counts.

```
pos = np.arange(len(mydict))
plt.bar(pos, [x[1] for x in mydict.iteritems()])
plt.xticks(pos, [x[0] for x in mydict.iteritems
    ()])
plt.show()
```

Multiple plots

```
fig = plt.figure()

f1 = fig.add_subplot(221)
data.hist(bins=25,color='navy',ax=f1)
f1.set_title('Histogram')
f1.set_xlabel('Xlabel')

f2 = fig.add_subplot(222)
f3 = fig.add_subplot(223)
f4 = fig.add_subplot(224)
```

Modules

- **Each module has its own namespace**. So if we do `import mymod`, then we can use items in mymod via `mymod.modfunc()`. It is possible to import all items from a module directly via `from mymod import *`, but this can lead to name mangling, and should be used with care.
- Any file "myfile.py" is a **module**, and can be imported by using `import myfile`.
- Can add code that is only called when the module is run as a main **script** (as `python module.py`):

```
if __name__ == "__main__":
    some_code_here
```

- Import a module from an **absolute path**:

```
import sys
import os
sys.path.append(os.path.expanduser('~') + "/path
    /to/file")
```

- Import a module from a **relative path**:

```
  from ..pkg import module
```

imports module.py in the folder pkg, where pkg is in the parent directory of the current location.

- **NB:** if the `__init__.py` file is not present, then can't import module files from other folders, solution is to add this file (it may be left empty).

Packages

- A **package** is a directory with a collection of modules (files), plus an `__init__.py` file. Packages can be used in two ways, either write

```
import mypackage
mypackage.module.function()
```

where module corresponds to a file module.py, or use

```
from mypackage import module
module.function()
```

Pandas

- Basic functions for data frames

```
import pandas as pd
df=pd.read_csv("myfile.csv", header=None, sep=";
    ",names=["first",'c2']))
# toy dataframe
df = pd.DataFrame({"c1": [1,2,3,4], "c2":
    [1,1,2,2]})
df.shape
df.columns
df.dtypes
df.describe()
df.head(10)
```

- Filtering data frames

```
# indexing
df.ix[0,1]
# extract columns
df[["c1"]]
df.filter(items=["c1","c2"])
# extract rows
df.filter(items = [2,3],axis=0)
# select elements
df[df.c1 < 5]
df.query("c1 > 4")
```

So to get the fourth row, can do either `df.ix[3,:]` or `users.filter(items=[3],axis=0)`.

- For data series, can do

```
series[series>3].index
series.sort_values(ascending=True)
```

- Missing data: `df[df.c1.isnull()]`, can do `df = df[df.c1.notnull()]`.
- Data summaries

```
df.c1.hist(bins=20)
plt.show()
```

Split, apply, combine in Pandas

Grammar of data

Verb	Pandas	SQL
Query	query()	SELECT WHERE
Sort	sort()	ORDER BY
Projection	[[]]	SELECT COLUMN
Select-distinct	unique()	SELECT DISTINCT COLUMN
Assign	assign	ALTER/UPDATE
Aggregate	describe(), mean(), max()	COUNT(),AVG(), MAX(), SUM()
Sample	sample()	RAND()
Group-agg	agg,count,mean	GROUP BY
DELETE	drop	DELETE/WHERE

Numpy arrays

- Create an array using `array([])`. You always need a list directly within `array()`. For example `array([[1,2],[3,4]])` creates a matrix with rows equal to the inner lists.
- Create an array of zeros with a given size: `zeros((n1,n2))`.
- Array with elements of different lengths, do `array([[0,1],[1]],object)`, which creates a different type of array.
- Operations:
  - Use `dot(a,b)` for any type of matrix multiplication, note that vectors do not distinguish between row/column versions.
  - Use `add(a,b)` for adding arrays *elementwise* and `subtract(a,b)` for subtracting elementwise.
  - Note that these operations will work on lists, but they will output an array, so try to work consistently with arrays within a function.
- Converting from arrays:
  - Use `list()` to convert an array to a list. A matrix will become a list of arrays containing the rows.
  - Use `float()` to convert an array of length 1 to a floating-point number.
- Pitfalls:
  - Printing an array just returns the vector/matrix/... as a list, so something could be an array but look like a list.

Pickle

Saving objects, useful for dataframes etc.

```
# Save object
import cPickle
with open('pfile.pkl','wb') as f:
    cPickle.dump(object,f)

# Read saved object
with open('pfile.pkl','r') as f:
    object = cPickle.load(f)
```

Timing

```
import time
start = time.time()
print("hello")
end = time.time()
print(end - start)
```

Regexp in Python

```
## replace parts of a string
nstr = re.sub(pattern, replacement, str, max=0)
```

Pattern	Matches
^	Beginning of line
$	End of line
.	Exactly one character
[...]	Any one character in the brackets
[^...]	Any one character *not* in brackets
x*	0 or more times x
x+	1 or more times x
x?	0 or 1 times x
\w	Word characters
\W	Nonword characters
\s	Whitespace
\d	Digits. Equivalent to [0-9]