CSS Home Assignment & Python Tutorial

Claudia Wagner

University Koblenz-Landau
GESIS - Leibniz Institute for the Social Sciences
clwagner@uni-koblenz.de

April, 2014

Who am I?

- Post doctoral researcher at GESIS - Leibniz Institute for the Social Science
- Computational Social Science department (Data Science team)
- Current Research: Measuring, modeling and predicting collective and individual behavior on the Web
- PhD at Graz University of Technology "Emergent Structure, Usage and Semantics in Social Streams"

What happens in a CSS department?

- Example I: How do politicians communicate on Twitter? What are their conversational practices? How can we measure and model their practices?
- Example II: What's the nature and evolution of online food preferences? How can we measure online food preferences? How do online food preferences relate with offline food preferences?
- Example III: Measuring the self-focus bias of different countries when describing their cuisine in Wikipedia. Can we learn a global food ontology from Wikipedia which incorporates the local viewpoints of different countries? (e.g., a nutrient X might be a considered as breakfast in country W but not in country Y)
- If you are interested in master projects, thinking about doing a PhD or looking for SHK job, talk to me!

Before we start

Who are you?

Logistics I

- Two home assignments (max number of points per assignment might differ!)
- Groups of max. 2 people
- You need >51% of total number of points (make sure that you have at least 51% of each assignment)
- You need to be able to explain and present your solution in class
- Submit your solutions as ipython notebooks via email clwagner@uni-koblenz.de
- subject of email: "CSS2014 ass N " where N is 1 or 2
- filename: first 2 letter of firstname student1 %underline% lastname student1 %underline% first 2 letter of firstname student2 %underline% lastname student2 %underline% ass1
- example: clwagner_mastrohmaier_ass1.ipynb

Logistics II

- May 7th first home assignment (submission deadline: June 11th)
- June 18th second home assignment (submission deadline: July 14th)
- June 18th correction class of the first assignment (be present in class and prepared to explain what you did!)
- July 16th correction of the second assignment (be present in class and prepared to explain what you did!)

Logistics III

- If you have questions use our newsgroup: compsocsci at webnews.uni-koblenz.de
- Please check the newsgroup on a regular base! We will announce important changes/updates there!
- Help each other! But don't post your solutions before the correction class.
- You can get up to 4 extra points (2 for each assignment) when answering questions of your colleagues and/or providing useful, additional materials that will help you colleagues to better understand the topics of the course.

Topics of Assignments

- First assignment: social network acquisition, representation and analysis
- Second assignment: social network dynamics and evolution

Questions

Questions about the logistics???

Why Python?

Python is:

- very readable
- easy to learn
- interpreted & interactive – like a UNIX shell, only better
- object-oriented – but not religious about it
- slower than C, but it is easy to integrate C, Fortran, Java

Installing Python

- Included in most distributions of Linux and Mac OS X
- Downloadable from `http://www.python.org/download/`
- Install ipython and notebook server: http://ipython.org/install.html
- Libraries:
  - pip: `https://pypi.python.org/pypi/pip`
  - virtualEnv: `https://pypi.python.org/pypi/virtualenv`
  - NumPy: `http://sourceforge.net/projects/numpy/`
  - matplotlib: `http://sourceforge.net/projects/matplotlib/files/matplotlib/`
- Editors (anything that knows how to handle tabs and spaces)
  - e.g., Eclipse, emacs, vim
  - ...

Invoking Python

- Interactive mode: Open a console/terminal window and run `python`
- Fancier alternative: run `ipython`
- Even more fancy: ipython notebook –notebook-dir=path
- Executing files: `python myfile.py`

Why will we use iPython and notebooks?

- IPython provides rich functionalities to help you make the most out of using Python
  - Dynamic object introspection. ?/?? gives access to access docstrings, function definition etc. (dir() lists all in-scope variables)
  - Tab completion
  - Complete system shell access. Lines starting with ! are passed directly to the system shell (e.g., !ls)
  - linemagic (e.g. %run script.py or %cd) and cellmagic (e.g., %%%writefile('file.py')) – %lsmagic lists all magic
  - ...
- A notebook is web-based and provides the same features but support for code, text, mathematical expressions, inline plots and other rich media.

iPython Notebooks

- `ipython notebook --ip=localhost --notebook-dir=/pyNotebooks/mydir --pylab inline`
- Not working? Maybe you need to add 127.0.0.1 to the "authorized websites" of you antivirus software
- Combine different cells (text, code, markup, ...)
- Execute all (code) cells
- Convert notebook to different formats
- `ipython nbconvert --to html yourNotebook.ipynb`

Language basics

Dynamically typed (no variable declarations)

```
n = 42                  # integer
x = 3.14159             # float
x = "Hello world!"      # string
```

String delimiters:

```
s = "Hello world!"
s = 'Hello world!'
s = """Hello
world!"""               # multi-line string
```

String Functions

```
text = "Hello world!"
len(text)  #12
text[0:3]  #'Hel'
text[3:] #'lo world!'
text.upper()  #'HELLO WORLD!'
text.find('l') #2
text.find('l', 2+1)  #3
```

Strings are immutable!

```
text[0] = 'J'  #TypeError
```

Booleans

- `True` is true, `False` is false
- 0, `""` and `None` are false, (almost) everything else is true

```
not A
A and B
A or B
```

Comparisons: ==, !=, <, <=, >, >=

```
2 == 2
1 < 2 <= 3
1 == 2 and "3" or "4"        # = "4"
```

Functions

```
def factorial(n):
    if(n==0):
        return 1
    else:
        recurse = factorial(n-1)
        res = n* recurse
        return res
```

- What happens if we call factorial(3)?
- What happens if we call factorial (1.5)?

Functions

The costs of flexibility and interactivity

```
def factorial(n):
    if(isinstance(n, int)):
     print "factorial is only defined for int"
return None
    if(n<0):
        return 1
    else:
        recurse = factorial(n-1)
        res = n* recurse
        return res
```

Control flow

```
if n == 0:
    print "n is 0"
elif n > 0:
    print "n is positive"
else:
    print "n is negative"
```

- Indentation matters!
- Don't mix tabs and spaces – configure your editor appropriately!

```
while n > 0:
    n -= 1
```

```
for n in [1, 2, 3, 4]:
    print n
```

Data Structures

- List: a sequence of items (mixed types possible). Indices are integer.
- Tuples: just like lists, but you can't change their values
- Dictionaries: a sequence of items (mixed types possible). Indices can be almost any type.
- Sets: unordered collection with no duplicate values.

Lists

- A string is a list of characters. Elements in a list can have different types.

```
firstList = [1, 2, "string", [8, 3]]
secondList = [5, 6]
mergedLists = firstList.append(secondList)
print mergedLists  #None
mergedLists = firstList + secondList
#[1, 2, 'string', [8, 3], [5, 6], 5, 6]
firstList.append(secondList)
print firstList
#[1, 2, 'string', [8, 3], [5, 6], 5, 6]
newList = [x*2 for x in secondList]
print newList #[10, 12]
```

- What happens if we call [x*2 for x in firstList]?

Lists

```
L = [1, 2, 3]
L[0]                    # = 1
L[1:3]                  # = [2, 3]
L[:-1]                  # = [1, 2]
L.append(4)             # -> [1, 2, 3, 4]
L += [5, 6]             # -> [1, 2, 3, 4, 5, 6]
del L[5]                # -> [1, 2, 3, 4, 5]
len(L)                  # = 5
L.reverse()             # -> [5, 4, 3, 2, 1]
L.sort()                # -> [1, 2, 3, 4, 5]
```

Aliasing

- Be careful with aliasing when you work with mutable objects
- Variables only hold references to lists (like to every other object)

```
firstList = [1, 2, "string", [8, 3]]
secondList = firstList
secondList[1] = 10
def remove_first(t):
    del t[0]

print firstList  #[1, 10, 'string', [8, 3]]
remove_first(secondList)
print firstList      #[10, 'string', [8, 3]]
```

Sets

- unordered collection, no duplicates
- supports set-theoretic operations like union or intersection

```
firstSet = set([1, 2, 3])
secondSet = set([1, 4, 8])
firstSet[0] = 1 # ERROR unordered, no indices
firstSet.add(5)
print(firstSet) #set([1, 2, 3, 5])

firstSet & secondSet # {1}
firstSet | secondSet # {1, 2, 3, 4, 8} union
firstSet - secondSet # {2, 3} difference
#symmetric difference
firstSet ^ secondSet # {2, 3, 4, 8}

immutableSet = frozenset([1, 2, 3])
newSet = [x*2 for x in immutableSet]
```

Dictionaries

a sequence of items (mixed types possible). Indices can be almost any type.

```
D = {"Mozart": 1756, "Schubert": 1797}
D["Mozart"]              # = 1756
D.get("Mozart")          # = 1756
D.keys()                 # = ['Schubert', 'Mozart']
D.values()               # = [1797, 1756]
D.update({"Beethoven": 1770})
D["Einstein"] = 1879
"Einstein" in D          # = True
len(D)                   # = 4
```

Tuples

Tuples are Immutable

```
t = ('a', 'b', '1')
t[0] = 'c'  #TypeError
```

zip function creates lists of tuples

```
a = range(3)
b = ('a', 'b', 'c')
c = zip(a, b)  #[(0, 'a'), (1, 'b'), (2, 'c')]
d = dict(c) # {0: 'a', 1: 'b', 2: 'c'}
```

Gather function arguments into a tuple

```
def printall(*args):
    print args

printall(1, 2, 3, 4)
```

Working with Lists, Dict, Set and Tuple

Iterate over a dictionary and format strings:

```
D = {"Mozart": 1756, "Schubert": 1797}
for name, year in D.iteritems():
    print "%s was born in %d" % (name, year)
```

List comprehensions:

```
quad = [x**2 for x in range(8)]
    # = [0, 1, 4, 9, 16, 25, 36, 49]
even = [x**2 for x in range(8) if x % 2 == 0]
    # = [0, 4, 16, 36]
```

Performance of Data Structures

- From a performance point of view tuples or frozensets are great because of their immutability. Python will know exactly how much memory to allocate for the data to be stored.
- When you need to store data that doesn't have to change think about using immutable data structures.
- Lists can be used as stacks or queues. It's easy to add or remove elements from the beginning or end (cf. pop or add).
- When you need fast lookup for your data, based on custom keys use a dictionary.

Performance of Lists

- indexing and assigning index is fast $\rightarrow$ O(1)
- append method is fast $\rightarrow$ O(1)
- pop() is fast if you remove last item of list $\rightarrow$ O(1)
- insert(i,item) $\rightarrow$ O(n)
- concatenation depends on the size k of the list that is being concatenated $\rightarrow$ O(k)
- But pop(i) is slow $\rightarrow$ O(n)
- sort() $\rightarrow$ O(n log(n))
- del operator $\rightarrow$ O(n)
- contains $\rightarrow$ O(n)
- iterate $\rightarrow$ O(n)

Example: Create a Lists

```
def test1(): # 6.5 ms
    l = []
    for i in range(1000):
        l = l + [i] # concatenation

def test2(): # 0.31 ms
    l = []
    for i in range(1000):
        l.append(i) # append

def test3(): # 0.15ms
    l = [i for i in range(1000)] # comprehension

def test4(): # 0.07ms
    l = list(range(1000)) # list range
```

Performance of Dictionary

- del operator $\rightarrow$ O(1)
- contains $\rightarrow$ O(1)
- set $\rightarrow$ O(1)
- get $\rightarrow$ O(1)
- copy $\rightarrow$ O(n)
- iterate $\rightarrow$ O(n)

Further useful data structures

- Counter (like table in R)

```
import collections
l = ['a', 'a', 'a', 'a', 'b', 'b']
collections.Counter(l)
#Counter({'a': 4, 'b': 2})
```

- OrderedDict is a dict that remembers the order that keys were first inserted
- Namedtuple: like tuple but with the ability to access fields by name instead of position index

Files

```
#read from csv file
import pandas as pd
fileUrl = 'https://myserver/rows.csv'
data = pd.read_csv(fileUrl)

#read/write json objects
import json
json.dump(x, file)
json.load(file)

# Write the text into a file
import csv
csvfile = open('file.csv', 'w')
csvfile.write(data)
csvfile.close()
```

Modules

Any Python file (e.g. `mymodule.py`) is a module and can be imported:

```
import mymodule

mymodule.myfunction()
```

```
from mymodule import myfunction
from mymodule import *    # rather discouraged

myfunction()
```

File-system directories can be used to create "namespaces", if they contain a file `__init__.py` (which may contain initialization code):

```
import mydir.mymodule
from mydir.mymodule import myfunction
```

Numpy

numpy is a N-dimensional array package

```
import numpy as np
b = np.array([6, 7, 8])
type(b) #numpy.ndarray
b.shape() #(3,)

a = np.arange(15).reshape(3, 5)
# [[ 0  1  2  3  4]
# [ 5  6  7  8  9]
# [10 11 12 13 14]]

A = np.ones((3, 3), dtype="float")
# array([[ 1.,  1.,  1.],
#        [ 1.,  1.,  1.],
#        [ 1.,  1.,  1.]])
```

Numpy

```
B = np.random.random((3, 3))
#array([[ 0.27903882,  0.78077845,  0.72217985],
#       [ 0.05440674,  0.29332926,  0.80661467],
#       [ 0.89496267,  0.28572368,  0.13621656]])

A*B # elementwise product
np.dot(A, B) #matrix multiplication

from numpy.linalg import *
eig(A)
A.transpose()
np.trace(A) %sum along diagonals of the array
```

Matplotlib

`matplotlib` is a plotting modul. `pylab` provides a procedural interface to the matplotlib object-oriented plotting library

```
from pylab import *
X = np.random.normal(0,1, 1024)
Y = np.random.normal(0,1, 1024)
scatter(X,Y)
show()
hist(X)
show()
```

Scipy

`scipy` is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python

- Provides many sub-packages e.g. `stats` for statistical distributions and functions, `sparse` for matrices and associated routines or `cluster` for clustering algorithms
- Tutorials: `http://docs.scipy.org/doc/scipy/reference/tutorial/index.html`

Statistics in Python

`stats` is a sub-package of scipy

```
#draw 100 samples from a normal distr
d = numpy.random.normal(size=100,scale=0.5,loc=2)
n,min_max,mean,var,skew,kurt = stats.describe(d)
#n  100
#min_max (0.42693119052258655, 3.513506785479902)
#skew -0.0189237657984
#kurt 0.266427349951
# mean 2.02938611556
```

Statistics in Python

- `stats` is a sub-package of scipy
- `statsmodels` and `pandas` for R-like data structures and statistical models

```
import numpy as np
from scipy import stats
import pylab
x = np.array([1, 2, 5, 7, 10, 15])
y = np.array([2, 6, 7, 9, 14, 19])
slope,intercept,r_value,p_value,slope_std_error
= stats.linregress(x, y)
predict_y = intercept + slope * x
pylab.plot(x, y, 'o')
pylab.plot(x, predict_y, 'k-')
pylab.show()
```

Distribution in Python

numpy.random (or scipy.random) allows to draw samples from several distributions (e.g., normal, gamma, binomial, pareto, zipf, lognormal)

```
# Build a vector of 10000 normal deviates
# with variance 0.5 and mean 2
mu, sigma = 2, 0.5
v = numpy.random.normal(mu,sigma,10000)
# Plot a normalized histogram with 50 bins
pylab.hist(v, bins=50, normed=1)
pylab.show()
```

Pandas

pandas provides fast, flexible, and expressive data structures designed to make working with relational or labeled data both easy and intuitive.

Series (similar to numpy ndarray):

```
import pandas as pd
s = pd.Series([5, 1, 2], index=['a', 'b', 'c'])
d = {'a' : 5, 'b' : 1, 'c' : 2}
# a    5
# b    1
# c    2
s["a"]
# 5
s[1:]
# b   1
# c   2
```

Pandas: Data Frames

```
d = pd.DataFrame({'one':s, "two":{"a":0, "b":3}})
#    one  two
# a    5    0
# b    1    1
# c    2   NaN
d['one'] # select column
d.loc['a'] # select row by name
d[1:3] # slice by rows
del d['one'] # delete column
```

DataFrame objects automatically align on both the columns and the index (row labels). Resulting object will have the union of the column and row labels.

Pandas: Splitting and Aggregating

```
d = pd.DataFrame({'one':s,
"two":{"a":"foo", "b":"foo", "c": "bar"}})
#    one    two
# a     5      foo
# b     1      foo
# c     2      bar

d.groupby("two")
```

Pandas: Splitting and Aggregating

```
grouped = d.groupby("two")
grouped.groups
# {'bar':[2], 'foo':[5, 1]}
grouped.aggregate(np.sum)
#  two
#  bar      2
#  foo      6
grouped.size()
#  two
#  bar      1
#  foo      2
```

Much more is possible: groups multiple columns, apply serveral aggregate function to one column (e.g., np.mean, np.std, pd.Series.nunique), define you own aggregation function, etc.

Regular expressions

`re` is a module for handling regular expressions.

```
import re

re.findall('[A-Za-z]+', 'Hello world!')
# = ['Hello', 'world']
```

*	0 or more	`findall(pattern, string)`
+	1 or more	`match(pattern, string)`
?	0 or 1	`sub(pattern, repl, string)`
[...]	certain characters	`split(pattern, string)`
[^...]	characters excluded	...

Pre-compiling regular expressions:

```
WORD_RE = re.compile('[A-Za-z]+')
WORD_RE.findall('Hello world!')
```

NetworkX

`networkx` is a module for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

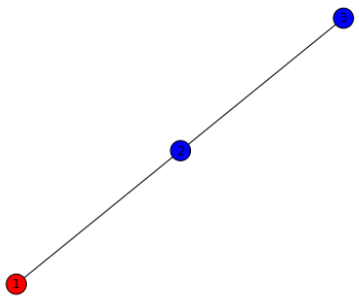
```
import networkx as nx

G = nx.Graph()
G.add_node(1)
G.add_edge(1, 2)
G.add_edge(2, 3)
```

Plot graphs:

```
import matplotlib.pyplot as plt
plt.figure()
nx.draw(G)
plt.savefig('graph.png')
```

NetworkX: Customized visualization

```
plt.figure()
plt.axis('off')
pos = nx.spring_layout(G, iterations=50)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_nodes(G, pos, [1],
    node_color='r')
nx.draw_networkx_nodes(G, pos, [2, 3],
    node_color='b')
nx.draw_networkx_labels(G, pos)
plt.savefig('graph.png', dpi=72)
```



NetworkX: Analyze your network

See `http://networkx.lanl.gov/reference/algorithms.html`

```
nx.info(G)
#Number of nodes: 3
#Number of edges:
#Average degree:   1.3333
nx.degree_centrality(G)
# {1: 0.5, 2: 1.0, 3: 0.5}
# fraction of nodes a node is connected with
```

NetworkX: Bipartite networks

```
B = nx.Graph()
B.add_nodes_from([1, 2, 3, 4], bipartite=0)
B.add_nodes_from(["A", "B", "C"], bipartite=1)
B.add_edges_from([(1, "A"), (2, "A"), (1, "C"), (3,
nx.draw(B)

# fold network
U = nx.bipartite.projected_graph(B,[1,2,3,4])
nx.draw(U)
```

Further reading

- Official Python tutorial: `http://docs.python.org/tutorial/`
- Python tutorial at the University of Toronto: `http://www.cs.toronto.edu/~gpenn/csc401/401_python_web/`
- Python 2.6 Quick Reference: `http://rgruet.free.fr/PQR26/PQR2.6.html`
- `http://docs.python.org/library/re.html`
- `http://docs.python.org/library/urllib2.html`
- `http://docs.python.org/library/xml.dom.html`
- `http://code.google.com/p/simplejson/`
- NetworkX: `http://networkx.lanl.gov/contents.html`

AWK

is great for processing column-oriented text data, such as tables.

```
 # print second and third column of file
awk < file.csv  '{ print $2,  $3 }'
# change delimiter from
# default whitespace to comma
awk < file.csv -F"," '{ print $2,  $3 }'

# $0 holds the entire current input line
# $NF holds last field

# awk is weakly typed
awk < file.csv '{print $2/10}'
# concatenate strings
awk < file.txt '{print $2+$3}'
```

AWK

```
 # how many users are friends of user 0
awk < Gowalla_edges.txt
'$1==0 {count +=1}
END {print count}'

# print sorted list of friends
awk < Gowalla_edges.txt
'$1==0 {print $1 $2}' |
sort --key=2

#regular expressions to filter rows
awk ' /^test/ { print $2 }'
```