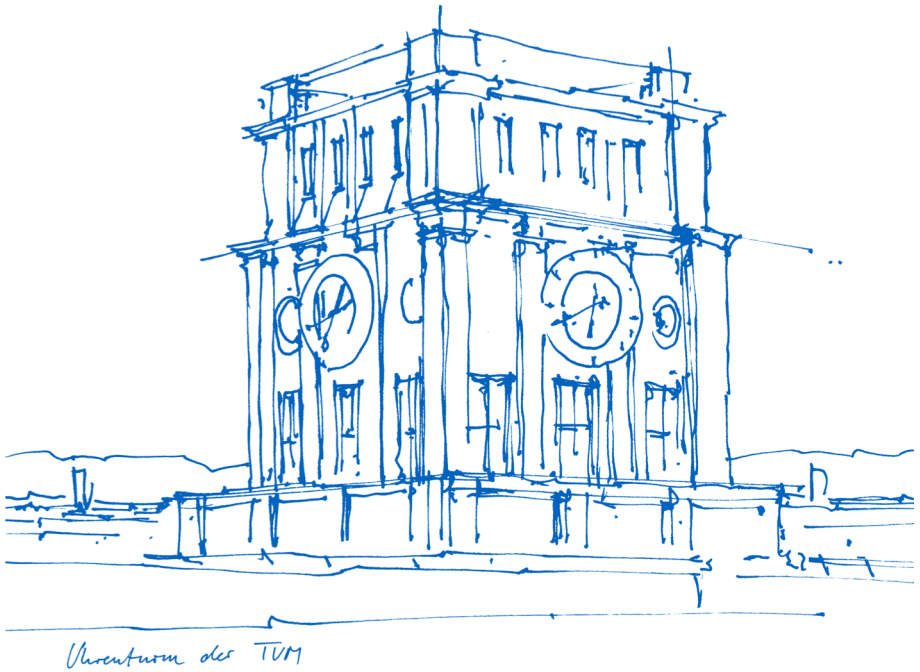
Exercises for
Social Gaming and
Social Computing
(IN2241 + IN0040)

Introduction to Exercise Sheet 1 - Introduction to Python and Network Visualization



Repetition: Python and IPython Books

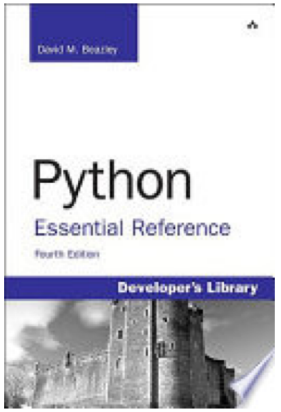
- **Learning Python:**

Python Essential Reference (2012)

by David M. Beazley, Safari Books

(especially **chapter 1: A Tutorial Introduction (25 pages)**)

free eAccess: https://eaccess.ub.tum.de/login



- **Learning IPython / Reference for IPython:**

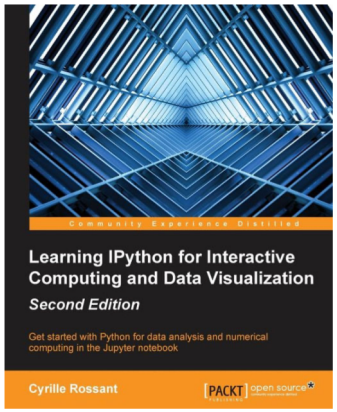
Learning IPython for Interactive Computing and Data Visualization (SECOND EDITION) by Cyrille Rossant, 175 pages, Packt Publishing, October 25 2015

(Especially (free) chapter 1.4. A crash course on Python)

free access:

http://nbviewer.ipython.org/github/ipython-books/minibook-2nd-code/blob/master/chapter1/14-python.ipynb

(do not try to open this ipynb with Jupyter directly. Instead, download all the ipynb's from the book from Github: https://github.com/ipython-books/minibook-2nd-code)



Installation

- get the installer from here: https://www.anaconda.com/download
- follow the installation instructions
- open Anaconda Prompt and type in:

```
conda update - -all
```

- type into Anaconda Prompt:

```
jupyter notebook
```

Exercise Sheet 1: Introduction to Python

- goal: get used to working with Jupyter Notebook and Python
- warm-up ex.: use loops to create a simple #-Pyramid:

- in the main ex. you will learn how to:
  - work with large datasets
  - choose the right format for your variables
  - choose the right format for your variables
  - use powerful tools to create, manipulate and display graphs

The Data: The Marvel Characters

- *MarvelAppearances.csv*:

  edges between character and comic issue appearance

Task

Task 1.1: Python Rhombus

a) Create a function which takes a number of levels (N) and **prints** a rhombus that looks like this for N = 4:

```
 #
###
###
 #
```

for N = 5:

```
  #
 ###
#####
 ###
  #
```

Hints:

- Do not forget the spaces left of the rhombus, except on the widest floor. In the example above 2 spaces to the left and at the peak.
- In order to execute a code cell, press Shift + Enter.

```
def printRhombus(N):
    #TODO: Create a Rhombus
printRhombus(5)
```

b) Extend the program by implementing user input. The user is asked to **enter a number** for the levels of the rhombus. Afterwards the pyramid is printed.

```
# TODO: Implement user input
num_levels =
printRhombus(num_levels)
```

Task

a) To get to see which characters appear together merge the Dataframe together with itself. This is not unlike the join operation in SQL.

```
# TODO: Merge charactes
# Hint: Use a left join

df_merged
```

b) Next we will group together the same characters on both sides to get the amount of times these characters appeared together. This will be used as the weights later.

```
df_merged2 = df_merged
# TODO: Group the Characters together

df_merged2
```

c) Now we are only interested in characters who have appeared at least 100 times together. Select those. Additionally delete rows of characters with references to themselves.

```
# TODO: Select the rows of characters with less than 100 appearances

# Delete these row indices from the dataframe
df_merged2.drop(indexNames, inplace=True)

# TODO: Select self-referential characters

# Delete these rows
df_merged2.drop(indexNames2, inplace=True)
df_merged2
```

Task

d) Now you have to **include your alter ego into the network**. Create a pandas Series with your name, and weights. Connect yourself to THANOS with weight 345.

```
# TODO:
# Create a series for your character who is connected to THANOS 345 times and add it to the dataframe

# TODO:
# Append the newly created series to the pandas data frame

# Create the graph from the dataframe
graph = nx.from_pandas_edgelist(df_merged3, source="Character_x", target="Character_y", edge_attr=True)
df_merged3
```

Task

e) Draw the resulting graph with the given options. Choose 2 layout [6] options that seem the most suitable for the data. Briefly discuss why you chose these over the others.

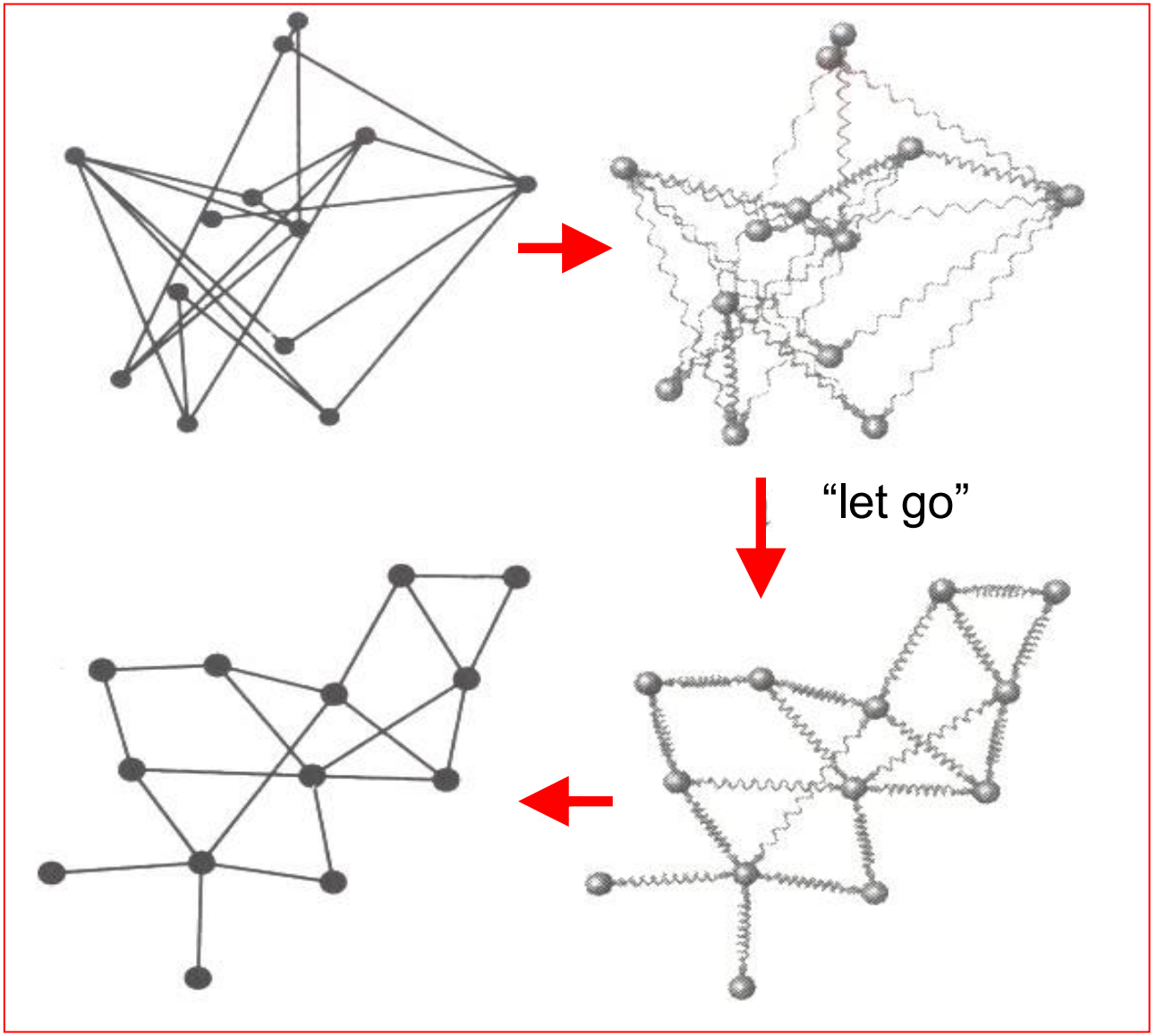
```
# Set the edge color according to the weight
edges,weights = zip(*nx.get_edge_attributes(graph,'size').items())

# Style the graph
options = {
    "font_size" : 14,
    "font_color" : '#552222',
    "node_color" : '#22FF22',
    "width" : 5.0,
    "edgelist" : edges,
    "edge_color" : weights,
    "edge_cmap" : plt.cm.Blues
}

plt.figure(1,figsize=(40,40))

# TODO: plot the graph

plt.show()
```

TODO: Write your observations here:

Drawing by Using Physical Analogies [5]

- Quality criteria used here:
  - nodes should spread well on the canvas
  - adjacent vertices should be close to each other
- Realized by:
  - Repelling forces between nodes: Charged nodes
  - Attractive + Repelling forces between nodes: Springs between nodes
  - → small deviations from uniform length edges

Drawing by Using Physical Analogies [5]



[5]

Drawing by Using Physical Analogies [5]

- → Edges will have different length in equilibrium. (Deciding whether a graph has an embedding in $\mathbb{R}^n$ with equal length edges (regardless of n) is NP-hard.)[5]

- Intuition: System will “move” into a state of minimal energy → Either minimize energy function or incrementally “move” nodes until “stable state” is reached.

Drawing by Using Physical Analogies [5]

- Orginal approach by Eades (see [5]): Change repelling and spring forces a bit:

$$\mathbf{F}_{repell}(\mathbf{v}_i, \mathbf{v}_j) = -\frac{c_\rho}{\| \mathbf{v}_i - \mathbf{v}_j \|^2} \frac{(\mathbf{v}_i - \mathbf{v}_j)}{\| \mathbf{v}_i - \mathbf{v}_j \|}$$

applied only to nodes not connected by edge of graph

$$\mathbf{F}_{spring}(\mathbf{v}_i, \mathbf{v}_j) = c_\sigma \log \frac{\| \mathbf{v}_i - \mathbf{v}_j \|}{l} \frac{(\mathbf{v}_i - \mathbf{v}_j)}{\| \mathbf{v}_i - \mathbf{v}_j \|}$$

applied only to nodes connected by edge of graph

log a/b instead of a-b exerts less force on far apart nodes

Drawing by Using Physical Analogies [5]

small constant

- Move node under total force F as: $\mathbf{v}_i \leftarrow \mathbf{v}_i + \delta \cdot F_{v_i}(t)$

whole spring embedder algoritm (see [5])

Input: Graph G=(V,E); inital placements $\{\mathbf{v}_i\}$ of nodes

Output: "Relaxed" new positions $\{\mathbf{v}_i\}$

```
for (t = 0; t < numberOfIterations; t++){
    forall (v_i ∈ V){
```

$$F_{v_i}(t) \leftarrow \sum_{\{u \mid (u,v_i) \notin E\}} F_{repell}(u, v_i) + \sum_{\{u \mid (u,v_i) \in E\}} F_{spring}(u, v_i)$$

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \delta \cdot F_{v_i}(t)$$

```
    }
}
```

Drawing by Using Physical Analogies [5]

- Modified approach by Fruchterman & Reingold (see [5]):

  1) Change repelling and spring forces a bit:

$$\mathbf{F}_{repell}(\mathbf{v}_i, \mathbf{v}_j) = -\frac{l^2}{\| \mathbf{v}_i - \mathbf{v}_j \|^2} \frac{(\mathbf{v}_i - \mathbf{v}_j)}{\| \mathbf{v}_i - \mathbf{v}_j \|}$$

$$\mathbf{F}_{spring}(\mathbf{v}_i, \mathbf{v}_j) = \frac{\| \mathbf{v}_i - \mathbf{v}_j \|^2}{l} \frac{(\mathbf{v}_i - \mathbf{v}_j)}{\| \mathbf{v}_i - \mathbf{v}_j \|}$$

$$\mathbf{F}_{repell} - \mathbf{F}_{spring} = 0$$

if

$$\| \mathbf{v}_i - \mathbf{v}_j \| = l$$

Drawing by Using Physical Analogies [5]

- Modified approach by Fruchterman & Reingold (see [5]):

  2) Speed up computation by restricting contributions to repulsive force on node v_i to "nearby" nodes. Use grid to determine which nodes are "nearby". Among these use only those repelling force contributions above a threshold

  3) Clip net displacement $\delta(t)$ on node at iteration t to prevent excessive displacements in late stages → better convergence

  4) Clip coordinates at canvas boundaries

Drawing by Using Physical Analogies [5]

- Further Modified approach by Frick et al. (see [5]):

  1) change forces a bit further in order to
     - slow down nodes with high degree
     - speed up computation further by dropping square-root computation

$$\mathbf{F}_{repell}(\mathbf{v}_i, \mathbf{v}_j) = -\frac{l^2}{\| \mathbf{v}_i - \mathbf{v}_j \|^2} (\mathbf{v}_i - \mathbf{v}_j)$$

$$\mathbf{F}_{spring}(\mathbf{v}_i, \mathbf{v}_j) = \frac{\| \mathbf{v}_i - \mathbf{v}_j \|^2}{l \ \Phi(v_j)} (\mathbf{v}_i - \mathbf{v}_j)$$

$$\Phi(v_j) = 1 + \frac{\deg(v_j)}{2}$$

Drawing by Using Physical Analogies [5]

- Further Modified approach by Frick et al. (see [5]):

  2) introduce additional “gravitational” force towards the “barycenter” to prevent disconnected components of graph to drift too far to the boundary of the canvas

$$\mathbf{F}_{grav}(\mathbf{v}_i, \mathbf{v}_j) = \gamma\, \Phi(v_j)\left(\frac{\zeta}{|V|} - \mathbf{v}_j\right)$$

Barycenter:

$$\zeta = \sum_{v_i \in V} \mathbf{v}_i$$

- Even simpler possibility: Drop Barycenter and direct gravitation towards canvas center ☺

$$\Phi(v_j) = 1 + \frac{\deg(v_j)}{2}$$

Drawing by Using Physical Analogies [5]

- Further Modified approach by Frick et al. (see [5]):

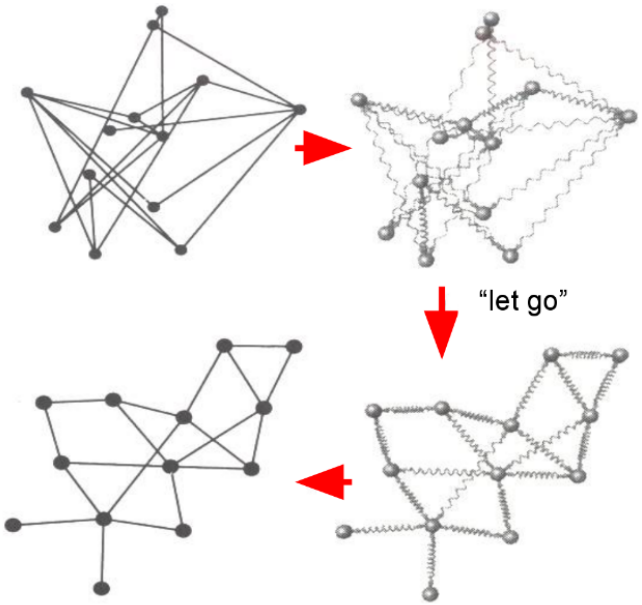
  3) Detect and suppress rotational and oscillatory moves by nodes (“ineffective” movements) (see [5] for details)

What we see: Spring embedder type of graph drawing algorithms are

- Robust towards adaptations
- Allow for an intuitive interpretation of change options

Problem 1.3: Graph Visualization and Analysis with IGraph Library

Fruchtermann & Reingold approach uses an analogy to classical mechanics for the graph modelling



$$\mathbf{F}_{repell}(\mathbf{v}_i, \mathbf{v}_j) = -\frac{l^2}{\| \mathbf{v}_i - \mathbf{v}_j \|^2} \frac{(\mathbf{v}_i - \mathbf{v}_j)}{\| \mathbf{v}_i - \mathbf{v}_j \|}$$

$$\mathbf{F}_{spring}(\mathbf{v}_i, \mathbf{v}_j) = \frac{\| \mathbf{v}_i - \mathbf{v}_j \|^2}{l} \frac{(\mathbf{v}_i - \mathbf{v}_j)}{\| \mathbf{v}_i - \mathbf{v}_j \|}$$

$$\mathbf{F}_{repell} - \mathbf{F}_{spring} = 0 \quad \text{if} \quad \| \mathbf{v}_i - \mathbf{v}_j \| = l$$

- By introducing **forces between the nodes**, we receive edges with equal length and few crossings
- Mechanical analogon: springs between the nodes

Further reading:

- http://igraph.org/r/doc/layout_with_fr.html
- Graph visualization slides on Moodle (just FYI, relevant for exam: just these exercise intro slides

Tips and Tricks

- for most TODOs it is sufficient to look at the pandas Manual and use libary functions

Submitting your solution

- work by expanding the .ipynb iPython notebook for the exercise that you downloaded from Moodle
- save your expanded .ipynb iPython notebook in your working directory
- submit your .ipynb iPython notebook via Moodle (nothing else)
- remember: working in groups is not permitted. Each student must submit their own .ipynb notebook!
- we check for plagiarism. Each detected case will be graded with 5.0 for the whole exercise
- deadline: check Moodle

Citations

(1) [Beazley 2013) David Beazley: Python Essential Reference, Safari Books 2013, E-Book available via www.ub.tum.de
(2) [Rossant 2015] Learning IPython for Interactive Computing and Data Visualization (SECOND EDITION) by Cyrille Rossant, 175 pages Packt Publishing, October 2015