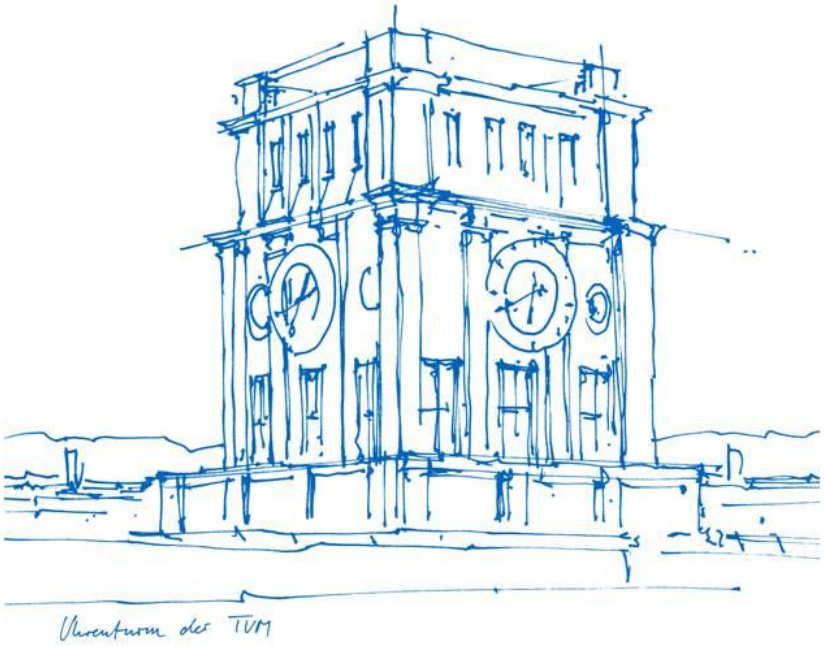
Exercises for
Social Gaming and
Social Computing
(IN2241 + IN0040)



Exercise Organization

- 1SWS of exercise → 6 worksheets:
  implementing and deepening lecture concepts;
  Form: IPhython notebooks
- Individal submission for each sheet: via Moodle (work in groups: not allowed!)
- Introductions to sheets via video
- all materials (slides, ipynb file to work on, data etc.): on Moodle
- Discussion of your solutions: in Q&A session Wednesdays, 14:15 on BBB

Exercise Grading

- 6 **individual** submissions of each student: each sheet is scored with a grade $m_i \in \{1.0, 2.0, 3.0, 4.0, 5.0\}$ (no 0.3 steps)

- $1/6\ (m_1+m_2+m_3+m_4+m_5+m_6)$ = exercise mark („freiwillige benotete Midtermleistung“)

- Grading Schema for Social Gaming (IN0040) and Social Computing (IN2241) Lecture Modules (5 ECTS) :

  - the better of:
    - grade of written exam with 0.3 bonus,
      if total grade of all exercise submissions is better or equal than 2.0
    - 50 % grade of written exam + 50 % total grade of all exercise submissions
      if total grade of all exercise submissions is better or equal 4.0 and better than the exam grade

  - grade of written exam,
    if exercises are not submitted or
    total grade of all exercise submissions is worse than 4.0 or
    total grade of all exercise submissions is worse than 2.0 and does not improve exam grade

Exercise Grading

PLEASE REGARD: In our Moodle: POINTS MEAN GRADES!!!

on Moodle, no actual „grades“ can be given, only „points“

→ if you get „3 points“ on excercise 2 that means:
„you got a grade of 3.0 on exercise 2“ or
„your exercise 2 submission was satisfatory but not great“
→ if you get „1 points“ on excercise 4 that means:
„you got a grade of 1.0 on exercise 4“ or
„your exercise 4 submission was very good“

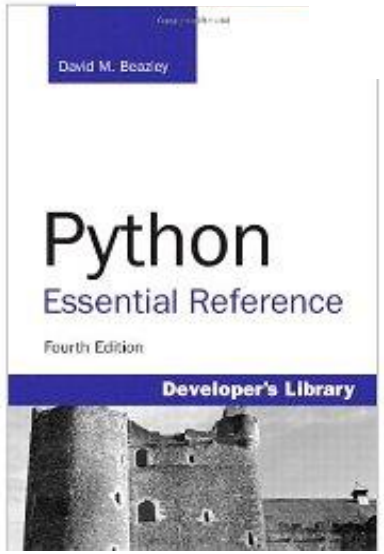
Exercise Content

Sheet Number	Exercise	Working Time
1	• Introduction to Python and Network Visualization	May 24-30
2	• Centrality Measures	May 31 – June 6
3	• Finding Groups & Clustering Methods	June 7 – 13
4	• Predicting Social Tie Strength with Linear Regression	June 14 - 20
5	• Natural Language Processing: Hate Speech detection + Social Context Influence	June 21 - 27
6	• Natural Language Processing: Modern Machine Learning Methods and Explainable AI	June 28 – July 4

Learning Python: Python and IPython Books

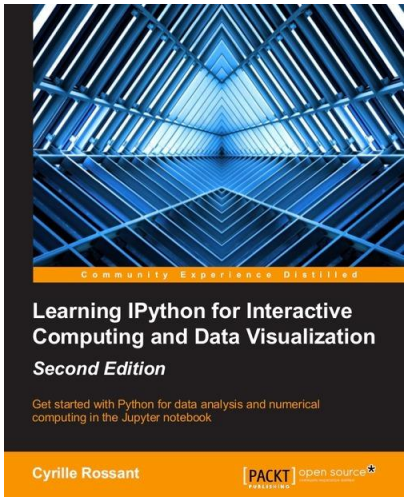
1 **Learning Python:**
Python Essential Reference (2012)
by David M. Beazley, Safari Books
(especially **chapter 1: A Tutorial Introduction (25 pages)**)

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



2 **Learning Python using 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)

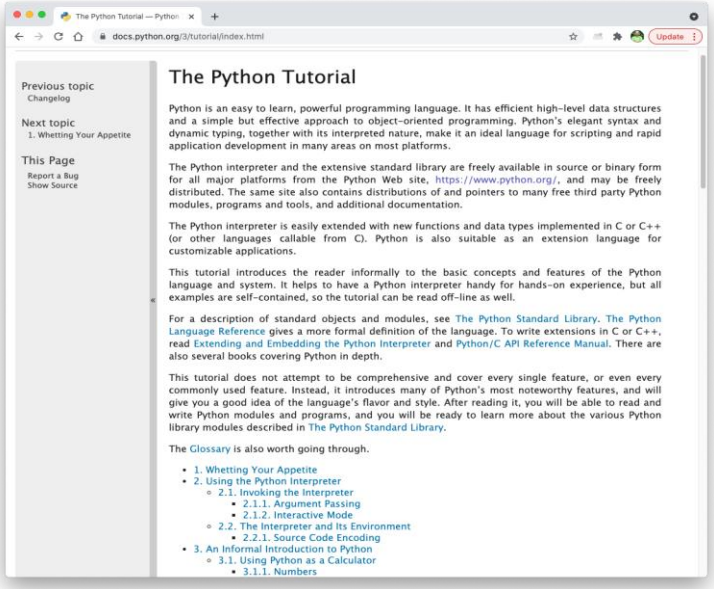


Learning Python: „Official“ Python Tutorial

3

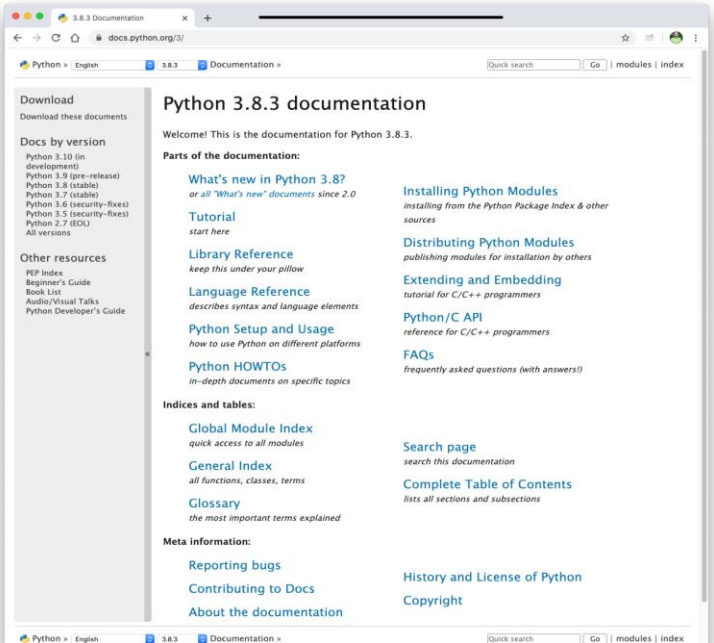
The „offical“ Python tutorial
can be found here:

https://docs.python.org/3/tutorial/index.html



Other „official“ Python documentation
(e.g. API of standard library)

https://docs.python.org



Python Environment Installation using Anaconda

- Follow instructions in Anaconda_Installation_Guide on Moodle:
- Install Anaconda
  - From Anaconda website
  - Choose your own OS

Python Environment Installation using Anaconda

- Now, we will create a new virtual environment to only contain the packages you will use in this course, and so that we can specify which versions of packages we want excactly.
  - Open Anaconda Prompt / Terminal
  - Download environment.yaml from Moodle
  - In Anaconda Prompt / Terminal, navigate to the folder containing environment.yaml
  - Run conda env create –f environment.yml

Running using Anaconda

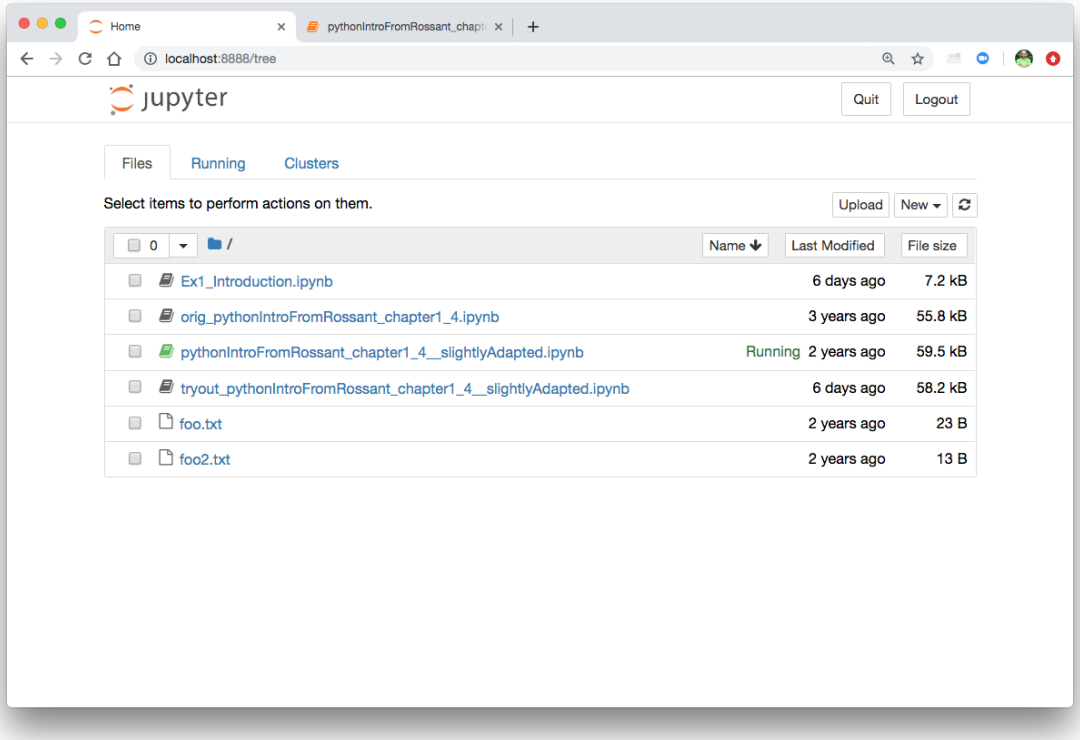
- Open Anaconda Prompt
- You can see your active environment in parentheses:

```
(base) C:\Users\rovod>
```

  - Before we run our code, we need to activate our environment
    - Run conda activate <yourenvname>
    - (If you are unsure about the name of your environment, run conda info --envs)
    - (If you forget to activate this environment (e.g. run the notebooks from base), the packages from base will be used, which are of different distribution than we need.)
- Navigate to the folder containing the exercises
- Run jupyter notebook
- Your web browser should open the notebooks.

Running using Anaconda

- Your web browser should now run the notebooks. It should look something like this:



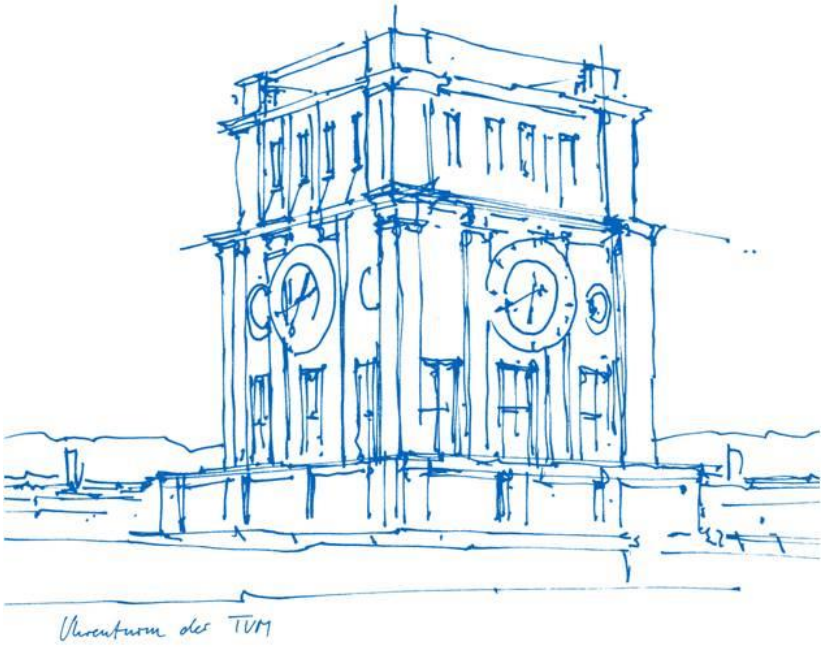
Check Moodle for a step-by-step installation guide!

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 submission

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

Exercise Sheet 1 - Introduction to Python and Network Visualization



Exercise Sheet 1: Introduction to Python and Network Visualization

- goal: get used to working with Jupyter Notebook and Python
- warm-up exercise: code sparse vector representation in Python to get used to the programming language
- in the main exercise you will learn how to:
  - work with large datasets
  - choose the right format for your variables
  - use powerful tools to create, manipulate and display graphs

The Data: The Twitch Streamers‘ Friends Network

- *musae_DE_target.csv*
  - each vertex represents a Twitch channel and contains multiple informations about the channel
- *musae_DE_edges.csv*
  - edges between the channels/streamers representing their friendship connection
  - Represented by `from` & `to` nodes, but are undirected as Twitch‘s friendship system is undirected
- *TwitchIDList.csv*
  - streamer‘s names for the used IDs of the dataset

Tasks

Task 1.1: Sparse Vector Representation

To get you prepared for using Python in the upcoming exercise sheets we will introduce some of Python's most important features on the example of sparse vectors. The basic idea is that sparse vectors (vectors whose positions are mostly zero) can be represented more efficiently by only saving a list of tuples containing the valueable data's position and its data. For example (0,0,0,4,0,0,42) would be represented by [(3,4),(6,42)].

In [ ]:
```
#Your example data
ExampleVector1 = #TODO
ExampleVector2 = #TODO
```

a) As the first step **complete** the `sparseVectorTransform()` function so that it transforms the given vector to a list of tuples as stated in the example above. Create reasonable `ExampleVectors` in the block above to test your function.

Hint: You can use `enumerate()` in a for-loop.

In [ ]:
```
def sparseVectorTransform(vector):

    #TODO:
```

Tasks (cont.)

b) In Python, variables are not assigned to a fixed datatype in contrast to Java for example. As a result, lists, for example, may contain elements of different datatypes. Even a function can be stored in a variable or list. Below you can see `combineSVs()` which combines two of our sparse vectors in tuple representation into one by only considering positions that are set in both vectors. The data entries for the respective position are combined by using the passed function. As you can see, the function is handed over just like regular parameters in Python.

In [ ]:
```
def combineSVs(func,SV1,SV2):
    returnlist = []
    for element1 in SV1:
        for element2 in SV2:
            if element1[0] == element2[0]: returnlist.append( (element1[0], func(element1[1],element2[1])))
    return returnlist
```

Tasks (cont.)

b) In Python, variables are not assigned to a fixed datatype in contrast to Java for example. As a result, lists, for example, may contain elements of different datatypes. Even a function can be stored in a variable or list. Below you can see `combineSVs()` which combines two of our sparse vectors in tuple representation into one by only considering positions that are set in both vectors. The data entries for the respective position are combined by using the passed function. As you can see, the function is handed over just like regular parameters in Python.

In [ ]:
```
def combineSVs(func,SV1,SV2):
    returnlist = []
    for element1 in SV1:
        for element2 in SV2:
            if element1[0] == element2[0]: returnlist.append( (element1[0], func(element1[1],element2[1])))
    return returnlist
```

It is now your task to **define** such a combining function to hand over to the `combineSVs()`. As we will only know at runtime what datatypes are contained in the vector, this function first needs to check which datatypes the given parameters have. For this task we will allow the vectors to contain Integers, Strings and Functions. Depending on datatypes, the result for their combination should be:

- Integer & Integer => Add both Integers
- String & String => Concatenate both String sequences
- String & Integer => Add the Integer as character at the end of the String
- Integer & String => Multiply the String sequence by Integer.
- Function & (String||Integer) => The result of the Function with the String or Integer as its parameter

In [ ]:
```
def combineElements (element1,element2):
    types = ['type1','type2']

    #TODO: Assign the datatypes of the elements to types


    #TODO: Create the correct result regarding the datatypes in types and return it


    #If the elements' datatypes do not fit any of our combination this exception will be thrown
    raise TypeError("Zulässige Komibnationen der Datentypen nicht eingehalten!")
```

Tasks (cont.)

c) The vectors below contain all possible combinations specified above. Use them to **compute** your final combined sparse vector in tuple representation. Then shortly **explain** in your own words the chances and pitfalls of Python's datatype independent variables. Three sentences will be sufficient.

```
v1 = (1,2,'Hallo Welt',len)
v2 = (3,'Hello world',4,'123')
print(combineSVs(combineElements, sparseVectorTransform(v1), sparseVectorTransform((v2))))
```

TODO: Write your explanation here:

Tasks (cont.)

Task 1.2: Twitch Social Network with NetworkX

For your second task you will now take your first steps into Social Computing / Social Gaming. We will therefore have a look at German streamers from Twitch [1]. Twitch is an international platform for live-streaming games or other content. The data set [2] has been created in 2019 and contains lots of information about different streamers. For our tasks we will only consider the German Twitch-streamers. Twitch offers lots of functionalities including a friendship system. The task is to draw a graph representing the **friendship network** of the hundred most watched German streamers.

In [ ]:
```
#Import the required libraries
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
```

Read in the data set by using the **Pandas** library [3]. It can represent the data sets as `DataFrame` objects and offers a multitude of functionalities for you. For your exercise sheets it will always come in handy to have a look at the used package's **documentations** online!

Inspect the three dataframes to fully understand what they contain. Don't confuse the Twitch IDs (`id`) with the node's IDs for our data set (`new_id`), which will form the key for our dataset.

Hint: You can have a look at the first entries of your Pandas data-frames with `Dataframe.head(10)`.

In [ ]:
```
#Contains a multitude of information about the streamers (adressed by their Twitch IDs).
nodes_DF = pd.read_csv('musae_DE_target.csv')
#Contains the friendship relations between streamers.
edges_DF = pd.read_csv('musae_DE_edges.csv')
#Contains the names of the top 100 Twitch-streamer for their Twitch IDs.
twitchNames_DF = pd.read_csv('TwitchIDList.csv')
```

Tasks (cont.)

Task 1.2: Twitch Social Network with NetworkX

For your second task you will now take your first steps into Social Computing / Social Gaming. We will therefore have a look at German streamers from Twitch [1]. Twitch is an international platform for live-streaming games or other content. The data set [2] has been created in 2019 and contains lots of information about different streamers. For our tasks we will only consider the German Twitch-streamers. Twitch offers lots of functionalities including a friendship system. The task is to draw a graph representing the **friendship network** of the hundred most watched German streamers.

In [ ]:
```
#Import the required libraries
import networkx as nx
import matplotlib.pyplot as plt
```

Read in the data set by using the **Pandas** library [3]. For a quick overview, you can refer to [4]. Pandas provides a powerful data structure called the "DataFrame" which is widely used in data science and machine learning. For your exercise sheets it will always come in handy to have a look at the used package's **documentations** online!

Inspect the three dataframes to fully understand what they contain. Don't confuse the Twitch IDs (`id`) with the node's IDs for our data set (`new_id`), which will form the key for our dataset.

Hint: You can have a look at the first entries of your Pandas data-frames with `Dataframe.head(10)` .

In [ ]:
```
#Contains a multitude of information about the streamers (adressed by their Twitch IDs).
nodes_DF = pd.read_csv('musae_DE_target.csv')
#Contains the friendship relations between streamers.
edges_DF = pd.read_csv('musae_DE_edges.csv')
#Contains the names of the top 100 Twitch-streamer for their Twitch IDs.
twitchNames_DF = pd.read_csv('TwitchIDList.csv')
```

a) First, we will deal with our graph's nodes, the streamers. We are only interested in streamers who have been streaming for more than 2000 days. **Filter** the data to contain only such rows. There are some columns we will not need later. **Remove** the unnecessary information by **dropping** the `id` , `days` and `mature` columns from `nodes_DF` . Then, to limit our data on the 100 most viewed streamers **order** `nodes_DF` by `views` and only **keep** the first 100 entries as `orderedNodes_DF` .

In [ ]:
```
#TODO:
```

Tasks (cont.)

b) Now we need to work on the friendship data-frame, our graph's edges. As we will only draw the top 100 streamers as nodes, we are only interested in friendship relations in between these 100 nodes. The relations are represented as "from"->"to" even though the friendship relation is considered undirected.

Fill `drop_indices` with all entry's indices that have their target outside of our 100 nodes. Therefore go through all entries of `edges_DF` and check if their `to` value is contained in `nodesList`. Then **drop** all the respective entries from our data-frame.

Hint: We do not need to worry about the `from` values as our merging process will take care of that part in the next task.

In [ ]:
```
drop_indices = []
nodesList = orderedNodes_DF['new_id'].tolist()

#TODO:
```

Tasks (cont.)

c) For the next step, we now have to **merge** our dataframes. This works similar to a SQL Join operation. We want to use an **'inner' merge** (similar to an "inner"=="normal" join in SQL) and make sure the nodes ID matches the friendship relation's origin. Thus it will only consider any friendships of our 100 nodes.

In [ ]:
```
merged_DF = #TODO
```

Now networkX [4] **reads** all the edges from our dataframe and creates a Graph object. **Take a look** at the number of nodes in your graph and **explain** your observations shortly.

In [ ]:
```
graph = nx.from_pandas_edgelist(merged_DF, "new_id", "to")

#TODO: Take a look at the number of nodes
```

TODO: Write your explanation here:

Tasks (cont.)

d) Before we finally draw our graph, lets add some **additional information** to it. We want to represent if the respective streamer is an official Twitch-partner by the **color** of a node. The `partner` column in our dataframe will provide you with the partner-status.

All entries were represented by their ID until now. To make our graph more legible we want to name all nodes with the streamers' actual names that are contained in `twitchNames_DF`. You have now to **create two dictionaries** that map the nodes ID to its partner state and streamer's name.

After that the given code will create the color-map from your dictionary that will later be used to apply the colors to our graph. Then the graph object is created.

Hint: As `twitchNames_DF` only contains the most 100 viewed streamer's names you can use it as an indicator whether you have solved everything correctly.

In [ ]:
```
partnerState_dict = #TODO
twitchNames_dict = #TODO

colormap = []
for node in graph:
    if(partnerState_dict[node] == True):
        colormap.append('Purple')
    else:
        colormap.append('Blue')

graph = nx.relabel_nodes(graph, twitchNames_dict)
```

Tasks (cont.)

e) Now to conclude your first exercise sheet you can finally draw the graph. NetworkX offers you multiple pre-defined graph layouts [5] that you can use. Don't forget to make use of the `colormap` and the set options! Matplotlib will then print your graph.

Try out the different layouts and chose two that fit the graph well. Then **explain** why they fit well and what **observations** you could draw from our visualized network shortly!

In [ ]:
```
plt.figure(1,figsize=(40,40))
options = {
    "font_size" : 30,
    "edge_color": 'lightGrey'
    }

#TODO: Draw the network


plt.show()
```

TODO: Write your explanation and observations here:

Tips and Tricks

- for most TODOs it is sufficient to look at the pandas manual and use pandas library functions

- You can get a better overview of a dataframe by printing it