# IN4080 – 2023
# Exercise set 1

Friday 25 August 2023

You will probably not manage to work through all of this exercise set during the group session. Continue to work on it by yourself after the group session and return to the teacher in later group sessions if you have any questions. This exercise set provides you with the tools you need in the mandatory assignments.

## Part 0: Get help to set up a working environment on your PC

Follow the installation instructions on the course web page to set up a working environment on your own computer. Alternatively, you can also work on the IFI cluster, where everything should be ready.

The following exercises should be solved interactively in Python. We recommend using a Jupyter Notebook, but you can also work in a standard interactive Python prompt.

## Part 1: Preprocessing data with NLTK

We will work with the book *Peter Pan*, which is available as plain text on Project Gutenberg. Download the *Plain Text UTF-8 file* from https://www.gutenberg.org/ebooks/16 and place it in your working directory. Rename the file if necessary.

Open the file in a text editor. Are there any particularities in the document that you'll need to watch out when processing?

Open Jupyter Notebook and create a new notebook in the same folder where you have placed the Peter Pan file. A notebook is organized into cells, and there are essentially two types of cells: text (Markdown) cells and code cells. You can execute a code cell by clicking on the Play button or by pressing Ctrl+Enter. Create some text cells and fill a code cell with the following code to load the Peter Pan text:

```
f = open('peterpan.txt', 'r', encoding='utf-8-sig')
t = f.read()
len(t)
```

You should see a number in the output cell. What does the number tell you?

We would now like to split the document into a list of sentences. Let us use the NLTK sentence tokenizer for this. Write the following code in a new cell and execute it (this may take a while):

```
import nltk.tokenize as tok
sents = tok.sent_tokenize(t, language='english')
len(sents)
```

(Jupyter may ask you to install some NLTK packages first. Just follow the instructions and try again.) Does the number of sentences correspond to your expectations? Inspect the data, e.g. with the following commands:

```
print(sents[0])
print("====")
print(sents[1])
print("====")
```

It turns out that the initial text contains a lot of line breaks that make the processing harder than necessary. Let us just replace all line breaks by a space. Return to the cell where you read the document and add the following lines:

```
import re
t = re.sub(r'\n+', ' ', t)
print(t[:100])
```

The slightly cryptic expression in the second line is a *regular expression*. If you are not familiar with regular expressions, let your teachers know. We won't need them now, but they can come quite handy for a lot of text processing tasks, so it's worth investing some time in learning the basics…

Re-load the file and run it through the sentence tokenizer again. How do you judge the result? Have new errors been introduced?

In this exercise set, we want to study the frequency distributions of words in the text. To this end, we will need to tokenize the text, but sentence splitting is not all that important. Let us read the text again, with all the line breaks, and tokenize it directly using NLTK:

```
f = open('peterpan.txt', 'r', encoding='utf-8-sig')
t = f.read()
tokens = tok.word_tokenize(t, language='english')
print(tokens[:30])
```

What happened to the line breaks? How many tokens does the text contain?

We will not do case folding in this exercise. Let us move on now and do some counting…

## Part 2: Frequency distributions and Pandas

Python provides a Counter class that counts repeated elements in a list and stores the counts as a dictionary. Let us use this class to count the tokens:

```
import collections
c = collections.Counter(tokens)
print(c.most_common(10))
print(c['with'])
```

You will see that punctuation signs are among the most frequent items in the counter. Remove them. Some hints:

- *string.punctuation* contains a list of punctuation symbols (you'll need to import *string* first)
- You can delete the 'and' item from the counter with *del c['and']*.

It would be nice if we could display the frequency distribution as a nicely formatted table. Let us use a DataFrame from the Pandas package for this. In general, it is easy to populate a DataFrame with the contents of a Python dictionary, as in the following code snippet:

```
import pandas as pd
d = {'apple': 5, 'orange': 8, 'banana': 51, 'strawberry': 20}
df = pd.DataFrame(d.items(), columns=['fruit', 'number'])
df.head()
```

You can do the same with your word counter.

How many types and how many tokens does the text contain? What is its type-token-ratio?

With Pandas, you can easily select rows of a dataframe according to a particular criterion. For example, this command displays all words that occur ten times or more:

```
df[df['count'] >= 10]
```

How many hapaxes are in the dataset? What percentage of all word types are hapaxes?

How many word types start with upper case A?

## Part 3: Plotting with Matplotlib

Matplotlib is a package for making plots and figures in Python. When using Jupyter notebooks, the plots are directly displayed in the notebook. This code snippet generates a simple plot:

```
import numpy as np
numbers = np.arange(10)
print(numbers)
import matplotlib.pyplot as plt
plt.plot(numbers)
```

Let us create a plot from our frequency distribution. You can directly use the plot() method of the dataframe as follows:

```
df = df.sort_values('count', ascending=False)
df.plot(x='word', y='count')
```

Does the result of this plot correspond to your expectations?

Modify the command to only display the 20 most frequent words. Also try to display all words on the x axis.

For frequency plots, it is more natural to use bar charts. You can easily switch to this type with the *kind='bar'* parameter.

Zipf's law states that the product of the frequency of a word and of its rank is approximately constant. Let us verify this law on a subset of our frequency distribution. First, select the 2000 first words of the sorted frequency table:

```
df_zipf = df[:2000].reset_index(drop=True)
```

This will also add a new (zero-based) index column that we can use for the rank. We can now add a new column with the product of rank and frequency:

```
df_zipf['z'] = df_zipf.index.values * df_zipf['count']
```

What are the highest, lowest and average values of z that you observe? Plot the z values as a line chart.

## Part 4: A different corpus in a different language

Browse the Project Gutenberg website and select another text in a different language that you understand. Apply all commands of the notebook again to this text, making sure that you use the correct language settings for the NLTK tools.

How do your observations differ from those made about Peter Pan? Which differences are due to the language, and which differences are due to the genre and length of the selected text?

## Extra exercises

If you are unfamiliar with Python, it might be worth to work through chapters 1 to 4 of the NLTK book.