Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

```
import pandas as pd # data wrangling
import jinja2 # template engine
import os # for file-related stuff

# create jinja env that can load template from filesystem
jinja_env = jinja2.Environment(loader = jinja2.FileSystemLoader)

df = pd.read_excel('Daten.xlsx')
template = jinja_env.get_template('template.xml')

with open('FertigesXML.xml','w') as output:
    output.write(template.render(data=df))
```

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

```
import sys
import collections
import pandas as pd
from tabulate import tabulate


file1 = pd.read_csv('file1.csv', sep=';', encoding='UTF-8')
file2 = pd.read_csv('file2.csv', sep=';', encoding='UTF-8')

columnnames1 = list(file1)
columnnames2 = list(file2)

if collections.Counter(columnnames1) == collections.Counter(columnnames2):
    print ("Number of columns and Names match, Comparison possible...\n\n")
else:
    print ("Number of columns and Names are not matching!!! Please check the input!")
    sys.exit('Error!')

# add suffixes to distinguish between actual and expected in the merger
file1 = file1.add_suffix('_e') # expected
file2 = file2.add_suffix('_t') # t


# merge them using the given key, use outer join
comparison = pd.merge(file1,file2, how='outer',
                      left_on=['Key_e'],
```

```
                    right_on=['Key_t'])

# create the columnwise comparison
for col in columnnames1:
    comparison[(col + '_c')] = comparison[(col + '_t')] == comparison[(col + '_e')]

# reorder the columns
comparison=comparison.reindex(sorted(comparison.columns),axis=1)

print(tabulate(comparison, tablefmt="pipe", headers="keys"))


# save the result as Excel file
comparison.to_excel('result.xlsx')

# names of the comparison column
check_colnames= [s + '_c' for s in columnnames1]

# initialize an empty dataframe for the log
logdf=pd.DataFrame(index=[True,False])

for column in check_colnames:
    t=comparison[column].value_counts() # returns a series
    tt=pd.DataFrame(t) # makes a DF out of the series
    logdf = logdf.join(tt,how='outer') # join the two dfs

# transpose for better readability
logdf = logdf.transpose()

# Ensure fixed sequence of the columns
logdf=logdf.reindex(sorted(logdf.columns),axis=1)

# write to disk
logdf.to_excel('logfile.xlsx')

# for better viewing on the screen
logdf.fillna('-',inplace=True)
pd.options.display.float_format = '{:,.0f}'.format

print(tabulate(logdf, tablefmt="pipe", headers="keys"))
```

Listing 1: fsafsdfs