Anhang

1.1 Quellcode

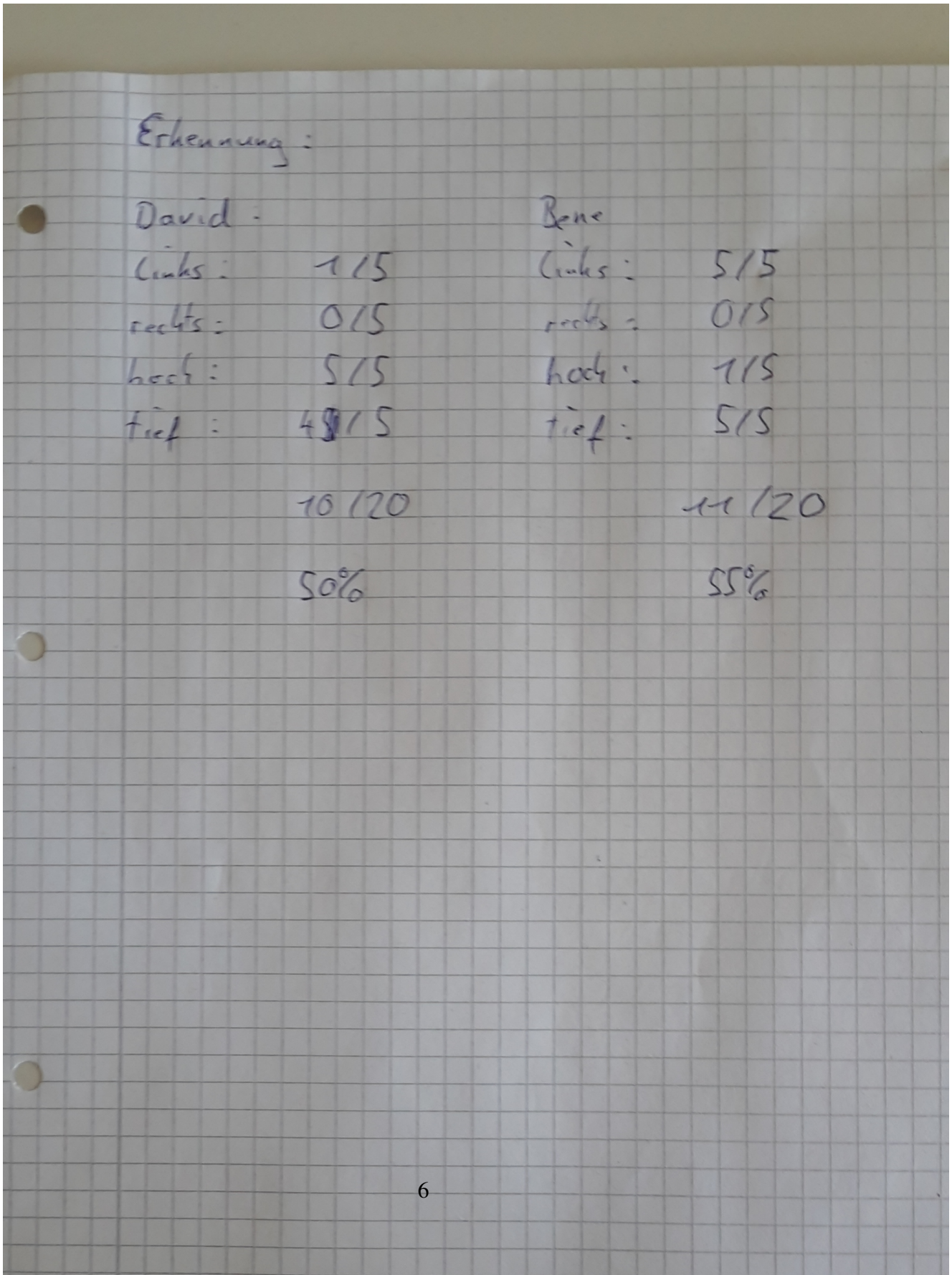
```
#-*-coding:utf-8-*-
"""
SpyderEditor

Thisisatemporaryscriptfile.
"""

import pyaudio
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal
import scipy.stats

FORMAT=pyaudio.paFloat32
SAMPLEFREQ=44100
FRAMESIZE=1024
NOFFRAMES=int(SAMPLEFREQ/FRAMESIZE)
WINDOW=512
def aufnahme(name):
    p=pyaudio.PyAudio()
    print('running')
    stream=p.open(format=FORMAT, channels=1, rate=SAMPLEFREQ,
    input=True, frames_per_buffer=FRAMESIZE)
    reg = True
```

```
    while reg:
        sample = stream.read(FRAMESIZE)
        sample = np.mean(np.fromstring(sample, 'Float32'))
        print(sample)
        if  sample > 0.005 or sample < -0.005:
            reg = False
    data=stream.read(NOFFRAMES*FRAMESIZE)
    decoded=np.fromstring(data,'Float32');
    stream.stop_stream()
    stream.close()
    p.terminate()
    print('done')
    plt.plot(decoded)
    plt.show()
    np.save(name,decoded)
    return decoded

def amplitude(file):
    file = np.load(file)
    spec = np.abs(np.fft.fft(file))
    fig, ax = plt.subplots(figsize=(800/100, 600/100), dpi=100)
    ax.plot(spec[:int(len(spec)/2)])
    ax.set_xlabel('frequency')
    ax.set_ylabel('amplitude')
    ax.set_title('Amplitudenspektrum')
    ax.autoscale(enable=True, axis='x', tight=True)
    #plt.show()

def windowing(file):
    counter = 0
    res = np.zeros(WINDOW)
    gauswin = scipy.signal.gaussian(WINDOW, WINDOW/4)
    cnt = 0
    while counter <= len(file):
        tmp = file[counter:counter+WINDOW]
```

```
            if len(tmp) is not WINDOW:
                vals = tmp
                tmp = np.zeros(WINDOW)
                for i in range(0,len(vals)):
                    tmp[i] = vals[i]
            tmp = np.multiply(tmp, gauswin)
            tmp = np.abs(np.fft.fft(tmp))
            res += tmp
            counter = counter + WINDOW/2
            cnt += 1
        res = np.divide(res, cnt)
        return res

def middle(name):
    value = np.zeros(len(np.load(name+str(1)+".npy")))
    counter = 0
    for i in range(0, 4):
        value = value + np.load(name+str(i+1)+".npy")
        counter = counter + 1
    value = np.divide(value, counter)
    plt.plot(value)
    np.save(name+"␣middle",value)

def korrelation(name):
    namearr = windowing(np.load(name+".npy"))
    hoch = windowing(np.load("hoch␣middle.npy"))
    tief = windowing(np.load("tief␣middle.npy"))
    links = windowing(np.load("links␣middle.npy"))
    rechts = windowing(np.load("rechts␣middle.npy"))
    corrList = []
    corrhoch = scipy.stats.pearsonr(namearr, hoch)[0]
    corrList.append(corrhoch)
    corrtief = scipy.stats.pearsonr(namearr, tief)[0]
    corrList.append(corrtief)
    corrlinks = scipy.stats.pearsonr(namearr, links)[0]
```

```
        corrList.append(corrlinks)
        corrrechts = scipy.stats.pearsonr(namearr, rechts)[0]
        corrList.append(corrrechts)
        correlation = max(corrList)
        if correlation == corrtief:
            return "tief"
        elif correlation == corrhoch:
            return "hoch"
        elif correlation == corrlinks:
            return "links"
        elif correlation == corrrechts:
            return "rechts"

#spec = windowing(decoded)
#amplitude(spec)

window = windowing(np.load('links␣middle.npy'))
fig, ax = plt.subplots(figsize=(800/100, 600/100), dpi=100)
ax.plot(window[:int(len(window)/2)])
ax.set_xlabel('frequency')
ax.set_ylabel('amplitude')
ax.set_title('Links')
ax.autoscale(enable=True, axis='x', tight=True)
```

1.2 Messergebnisse



Erkennung:

David:		Bene	
links:	1/5	links:	5/5
rechts:	0/5	rechts:	0/5
hoch:	5/5	hoch:	1/5
tief:	4/5	tief:	5/5
	10/20		11/20
	50%		55%