1.2

“Keep it stupid simple”

Meiner Meinung nach das wichtigste Prinzip der Programmierung. Mann sollte wenige Annahmen treffen über spezifische Komponenten im Program. Somit werden mit grosser Wahrscheinlichkeit komplizierte Refactorings vermieden.

“Liskov substitution principle”

Mit grosser Wahrscheinlichkeit ein VIP Prinzip der Objekt-Orientierten Programmierung. Funktionen einer Klasse, sollten auch dann gelten wenn eine Subklasse der respektiven Klasse angehört. Anders gesagt: Eine Elternklasse darf nicht spezialisierter sein als ihre Subklasse.

“Law of Demeter”

Das Prinzip besagt, dass mann so viel Information wie möglich “Verschwiegen” sollte von der restlichen Umgebung. Gilt meistens Funktionen und Variablen einer Klasse, Moduls, Header, etc.

1.3

Codefragment von einem Kollegen.
https://github.com/vc1492a/Hey-Waldo/blob/master/image_processing.py

```
import urllib.parse
import requests
from requests.auth import HTTPBasicAuth
from tqdm import tqdm
import os
from PIL import Image, ImageChops, ImageOps
import math
import numpy as np

# Please don't run this function unless you have to. There's a limit of 5000 requests per month. #

def bing_api(query_array, size_threshold, source_type, top, format):
    """Returns the decoded json response content

    :param query: query for search

    :param source_type: type for seacrh result

    :param top: number of search result

    :param format: format of search result

    A lot of this code shamelessly borrowed from: https://xyang.me/using-bing-search-api-in-python/

    """

    # Bing API key.

    API_KEY = "INSERT YOUR API KEY"

    for i in query_array:

        # set search url

        query = '%27' + urllib.parse.quote(i) + '%27'

        # web result only base url

        base_url = 'https://api.datamarket.azure.com/Bing/Search/' + source_type

        url = base_url + '?Query=' + query + '&$top=' + str(top) + '&$format=' + format

        # create credential for authentication

        user_agent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135
Safari/537.36"

        # create auth object

        auth = HTTPBasicAuth(API_KEY, API_KEY)

        # set headers
```

(Separation of concerns)
Solche Funktionen sollten lieber in einem separaten Modul definiert werden. In diesem Fall sehen wir einen API Aufruf in einem image processing Modul.

Niemals API-keys hardcoden.

```
        headers = {'User-Agent': user_agent}
        # get response from search url
        response_data = requests.get(url, headers=headers, auth=auth)
        # decode json response content
        json_result = response_data.json()

        # set the image counter to 0
        image_counter = 0
        # for the results in the json object
        for result in tqdm(range(0, top - 1)):
            # get the width of the image
            try:
                width = int(json_result['d']['results'][result]['Width'])
            except IndexError:
                print('Error with image.')
                continue
            # if width greater than threshold
            if width > size_threshold:
                # download the urls to the image url array
                image_url = json_result['d']['results'][result]['MediaUrl']
                image_counter += 1
                # open the source
                with open('raw-images/' + i + '-' + str(image_counter) + '.jpg', "wb") as file:
                    # get request
                    response = requests.get(image_url)
                    # write the file
                    file.write(response.content)

# bing_api(["Waldo"], 1024, 'Image', 50, 'json')

# make all the images square and of same size
def crop_and_size(input_file_path, output_file_path, dimensions):
    # create a directory if it does not exist
    if not os.path.exists(output_file_path):
        os.makedirs(output_file_path)
    counter = 1
    for image in os.listdir(input_file_path):
        if image != '.DS_Store':
            img = Image.open(input_file_path + '/' + image)
            cropped_and_sized = ImageOps.fit(img, dimensions, Image.ANTIALIAS)
            cropped_and_sized.save(output_file_path + '/' + str(counter) + '.jpg', 'JPEG')
            counter += 1
# crop_and_size('raw-images', 'cropped-and-resized', (1024, 1024))

# chops the images into smaller images for use
def chop(x_div, y_div, input_file_path, output_file_path):
    # create a directory if it does not exist
    if not os.path.exists(output_file_path):
        os.makedirs(output_file_path)
    counter = 1
    for image in tqdm(os.listdir(input_file_path)):
        if image != '.DS_Store':
            img = Image.open(input_file_path + '/' + image)
```

Funktion zu gross. Sollte in kleinere Subfunktionen aufgeteilt werden für bessere Lesbarkeit.

```
            (imageWidth, imageHeight) = img.size
            gridx = x_div
            gridy = y_div
            rangex = int(imageWidth / gridx)
            rangey = int(imageWidth / gridy)
            for x in range(rangex):
                for y in range(rangey):
                    bbox = (x * gridx, y * gridy, x * gridx + gridx, y * gridy + gridy)
                    slice_bit = img.crop(bbox)
                    slice_bit.save(output_file_path + '/' + str(counter) + '_' + str(x) + '_' + str(y) + '.jpg',
                                   optimize=True, bits=6)
            counter += 1

# chop(128, 128, 'cropped-and-resized', 'chopped-128')

# flips the images horizontally
def flip_horizontally(input_file_path, output_file_path):
    # create a directory if it does not exist
    if not os.path.exists(output_file_path):
        os.makedirs(output_file_path)
    counter = 1
    for image in tqdm(os.listdir(input_file_path)):
        if image != '.DS_Store':
            # open the image and transpose horizontally
            flipped = Image.open(input_file_path + '/' + image).transpose(Image.FLIP_LEFT_RIGHT)
            # save the image
            flipped.save(output_file_path + '/' + str(counter) + '-flip' + '.jpg', optimize=True, bits=6)
            counter += 1
# flip_horizontally('chopped', 'chopped-flipped')
# desaturates the images
def desaturate(input_file_path, output_file_path):
    # create a directory if it does not exist
    if not os.path.exists(output_file_path):
        os.makedirs(output_file_path)
    for image in tqdm(os.listdir(input_file_path)):
        if image != '.DS_Store':
            # open the image and convert to grayscale
            desaturated = Image.open(input_file_path + '/' + image).convert('1')
            # convert back to rgb
            desaturated = desaturated.convert('RGB')
            # save the image
            desaturated.save(output_file_path + '/' + image, optimize=True, bits=6)
# desaturate('chopped-64', 'chopped-64-gray')
# converts the images to black OR white
def black_or_white(input_file_path, output_file_path):
    # create a directory if it does not exist
    if not os.path.exists(output_file_path):
        os.makedirs(output_file_path)
    for image in tqdm(os.listdir(input_file_path)):
        if image != '.DS_Store':
            # open the image and convert to black or white
            desaturated = Image.open(input_file_path + '/' + image).convert('L')
            bw = np.asarray(desaturated).copy()
```

Nach jeder Function findet mann ein Kommentar wo zeigt wie mann die Funktion aufrufen kann. Funktionen sind somit lesbarer.

```
            # pixel range is 0...255, 256/2 = 128
            bw[bw < 128] = 0    # Black
            bw[bw >= 128] = 255     # White
            # get the image from the converted array
            imfile = Image.fromarray(bw)
            # convert back to rgb
            imfile = imfile.convert('RGB')
            # save the image
            imfile.save(output_file_path + '/' + image, optimize=True, bits=6)

# black_or_white('chopped-128', 'chopped-128-bw')
```
