# Data Set Preparation

Solve the following exercises and upload your solutions to Moodle (unless specified otherwise) until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are allowed to write additional functions, classes, etc. to improve readability and code quality.

## Exercise 1 – Submission: Image Data Set via NextCloud Server 50 Points

In this exercise, you will collect images along with their labels for the data set. For this, you have to take 50 pictures, e.g., with your cellphone and come up with appropriate labels. You can choose which scenes/objects you take pictures of, as long as you adhere to the following rules: The pictures must

- not exceed 250kB (=250 000 Bytes) per picture (a script for decreasing the size of an image is provided in the supplements),
- not show humans or parts of humans,
- not show personal/private information (license plates, door bell nameplate, private documents, etc.),
- be unique with a maximum of 5 pictures of the same object/scene,
- be JPEG files,
- have a minimum size of $100 \times 100$ pixels (other than that, there is no specific or consistent layout required),
- have the label as file name - if you have multiple instances of one label, just number the files incrementally (`cat.jpg, cat1.jpg, cat2.jpg, ...`),
- be taken by you (do not use images from the Internet or from other people without their explicit permission), and
- not include watermarks (make sure to deactivate watermarks if they are enabled on the device you use to take the pictures).

**Important note on submission:**

- Create a ZIP archive containing the 50 JPEG files + the CSV file that contains the labels (see below). The name of the ZIP file should be your student matriculation ID starting with `k` and followed by the 8-digit ID number (if necessary, include leading zeros, e.g., `k01234567.zip`).
- This exercise is *not* submitted via Moodle. Instead, a NextCloud server is provided which you must use to upload the ZIP file:

  `https://cloud.ml.jku.at/s/gMfAC3fNdfkDzkE`

  You need the following password: `MTrx53SPaT`. Please keep in mind that the server might be slow when a lot of students upload at the same time, so make sure to upload in time.

**Note on collected images:**

The JPEG files from all students will be pooled into one large data set, which we will use to train our model. This data set will be made available to all students in this course. Image metadata (GPS information, device type, etc.) will be removed before the data set is made available.

## Exercise 2 – Submission: `a1_ex2.py` 50 Points

Write a Python function

```
validate_images(input_dir: str, output_dir: str,
                log_file: str, formatter: str = "07d")
```

that validates images as follows:

- `input_dir` specifies the input directory where the function should look for files. The files must be searched recursively (i.e., including subdirectories) and then must be sorted in ascending order according to their file names (use the absolute paths for sorting). If `input_dir` is not an existing directory, a `ValueError` must be raised (with a message `input_dir` *is not an existing directory*).
- `output_dir` specifies the directory where all valid images (from the sorted list of files above) must be copied to. If the directory does not exist, it must be created. Image files are considered valid if the following rules are met:
  1. The file name ends with `.jpg`, `.JPG`, `.jpeg` or `.JPEG`.
  2. The file size does not exceed 250kB (=250 000 Bytes).
  3. The file can be read as image (i.e., the `PIL`/`pillow` module does not raise an exception when reading the file).
  4. The image data has a shape of `(H, W, 3)` with `H` (height) and `W` (width) larger than or equal to 100 pixels, and the three channels must be in the order RGB (red, green, blue). Alternatively, the image can also be grayscale and have a shape of only `(H, W)` with the same width and height restrictions.
  5. The image data has a variance larger than 0, i.e., there is not just one common pixel in the image data.
  6. The same image has not been copied already.

  The base name (without any extension) of the copied file must be as defined by the format string `formatter` (see below), which expects a single number that is used to apply the format string on. The number must be an integer starting at 0, and it is incremented by 1 for every file that has been copied. The extension of every copied file must be `.jpg`. Example:
  - Input files:
    * `cat.jpeg` (valid)
    * `dog2.png` (invalid)
    * `tree1.jpg` (valid)
    * `tree2.JPG` (valid)

  - Format string: `"07d"`
  - Output image files:
    * `0000000.jpg` (from `cat.jpeg`)
    * `0000001.jpg` (from `tree1.jpg`)
    * `0000002.jpg` (from `tree2.JPG`)

  In order to retain the label for each image, the new file name and the corresponding label have to be written to a `csv` file (file name: `labels.csv`), where the first column is the file name and the second column the corresponding label (headers: `name` and `label`), like this:

```
name;label
0000000.jpg;cat
0000001.jpg;tree
0000002.jpg;tree
0000003.jpg;car
0000004.jpg;cat
...
```

  Note that you must not include the numbering from the original file name in the csv file - if there is any (e.g. `cat1.jpg, cat2.jpg`). Please also note that you should use simple letter-only labels - no digits, no underscores etc. (e.g. dog, cat, desk, tree, ...).

- `log_file` specifies the path of the log file to which file names of invalid files must be written. The format of `log_file` must be as follows:
  - Each line must contain the file name of the invalid file, followed by a comma, an error code and a newline character.
  - The error code is an integer with 1 digit, corresponding to the list of rules for file validity from above (i.e., there are a total of 6 error codes). Only one error code per file must be written, and the rules must be checked in the ascending order 1, 2, 3, 4, 5, 6.
  - Each file name must only contain the relative file path starting from `input_dir` (this means `input_dir` itself is *not* part of the relative path anymore).

  For the example from above, the log file should contain the following line (with a trailing newline character):

```
dog2.png,1
```

  The log file is always newly created (even if it already exists), and potentially missing intermediate directories are also created.

- `formatter` specifies an optional format string to use when writing the output images. It expects a single number that is used to apply the format string on (e.g., with the default value of `"07d"`, the result should be the same as `"{:07d}".format(some_number)`). The result is the base name of the file, without any extension.

The function must return the number of valid files that were copied as integer. In the example above, 3 would be returned.

**Hints:**

- You can store the hashes of all copied files to check if the current file has already been copied.
- `os.path.getsize()` will report the size of a file.
- Loading an image from a file can be done with the external Pillow package (make sure to install it if necessary) and `PIL.Image.open()`.
- You can use the attribute `my_image.mode` to check the mode of an image (e.g., `"RGB"`).
- You can use the module `shutil` and then `shutil.copy()` to copy a file.
- `input_dir` might be an absolute or relative path. You can use `os.path.abspath()` to get the absolute path of `input_dir`.