Mandatory Assignment 4

Basic Python Programming (30/40 + 3 bonus points)

University of Oslo - IN3110/IN4110

Fall 2021

Your solution to this mandatory assignment needs to be placed in the directory `assignment4` in your Github repository. The repository has to contain a `README.md` file with information on how to run your scripts, required dependencies and packages (versions the code ran with) as well as how to install them. Documentation on how to run your tests[1] is required. Furthermore, your code needs to be well commented and documented. All functions need to have docstrings explaining what the function does, how it should be used, an explanation of the parameters and return value(s) (including types). We **highly** recommend you use a well-established docstring style such as the Google style docstrings[2]. However, you are free to choose your own docstring style. We expect your code to be well formatted and readable. Coding style and documentation will be part of the point evaluation for **all tasks** in this assignment.

Ifi Machines & User Privileges Since some compatibility issues might arise, when solving these tasks on different systems, we recommend you to checkout the remote login to the Ifi machines [3], when running into issues on your own machine. You can find an outline of the steps needed to set up the machines below[4]. We are aware of the limited user privileges. If you prefer to work with a GUI, you can also use the Ifi workstations remotely. [5] To test your setup on ifi machines, you can install a virtual environment.

On the IFI machine you can install `virtualenv` following the steps outlined below:

```
cd ~
pip3 install --user virtualenv
export PATH=$PATH:~/.local/bin
virtualenv -p python3 venv-in3110
```

[1] Not only important to get all of the credits, but it is a really important and not understated habit to be established :)

[2] `https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html`

[3] `https://www.mn.uio.no/ifi/tjenester/it/hjelp/it-vakten/laptophjelp/laptophjelp-guide/tilgang-til-uio-hjemmeomrade-og-ifi-linux-terminal/index.html`

[4] urlhttps://www.uio.no/studier/emner/matnat/ifi/IN3110/h21/getting-started.html

[5] `https://www.mn.uio.no/ifi/tjenester/it/hjelp/vdi/ifi-workstation-en.html`

and it will set up paths such that your script will be available system wide while the `venv` is active and you can `pip install`.

To activate the environment you can simply do:

```
source ~/venv-in3110/bin/activate
```

The you might need to upgrade pip and install some dependencies needed, e.g.

```
~/venv-in3110/bin/python -m pip install --upgrade pip
pip install opencv-contrib-python
pip install pytest
pip install numba
```

Once you installed everything you needed and you are done with the assignment, you can share the packages installed in your virtualenvironment, by adding `requirements.txt` file to you git repository.

To do so, you can simply

```
pip freeze > requirements.txt
```

A user or corrector can then create a `virtualenv` and install the exact preinstalled packages you used.

```
pip install -r requirements.txt
```

Note regarding the task in Section 4.3 The packages you develop are installed only for the venv named `venv-in3110`. Please do not add the venv folder to your repository. Check that the package works in non-edit mode, that is without the -e flag when pip installing, aka. check that you can install the package running the following command in the root instapy folder

```
pip install .
```

If you are still experiencing issues, it would be worth trying:

```
pip install . --user
```

How to share files between local and remote host

The commands shown here, are run on the local host, not in the `ssh`.

To copy files, e.g. your package into the remote host on ifi machines you can

```
scp path/to/local/file.ext username@afram.ifi.uio.no:path/to/
    remote/folder/
```

You need toinclude the `-r` flag if copying a folder.

For copying a file, `file.ext`, from remote host to local host you can

```
scp username@afram.ifi.uio.no:path/to/remote/file.ext path/to/
    local/folder
```

All correctors will have access to the ifi machines as well.

General File Summary *Summary Files to be created in this Assignment*

- `README.md` - a detailed information sheet (see above).
- `Files containing tests`

- `Files required and described in each subtask`

4.0 Profiling Warm-Up (5 points (IN4110 ONLY))

At some point in you career, you might encounter the issue that one of your programs you rely on on a daily basis is really, annoyingly slow. Since you are a great, efficient scientist or software-developer, that actually wants to make the deadline, you decide to make your program faster. The first step towards a fast implementation is figuring out, where your code spends its time. In this Warm-Up challenge you will go through the profiling techniques you learned in the lecture and apply them to some easy example code you can find in `test_slow_rectangle.py`.

Task:

- use manual timing to identify the slowest part of the code snippet

  - save the implementation using manual timing to `manual_timing_.py`

  - Hint: use `time.time()`

  - repeat the experiment at least 3 times

  - in `manual_report.txt` state the measured times and name the slowest part

- use the `timeit` module to verify the time measured manually for the slowest function

  - save your implementation using `timeit` to `timeit_timing_.py`

  - repeat the experiment at least 3 times

  - in `timeit_report.txt` state the measured times and say how they compare to manual timing

- use `cProfile` to compare the time measured for the slowest function to using `timeit` or manual timing

  - save your implementation using `cProfile` to `cProfile_timing_.py`

  - in `cProfile_report.txt` state the measured times and say how they compare to `timeit` or manual timing

4.1 Python for Instagram

In this assignment, you will make a program for turning your colorful image of choice into a dramatic grayscale or nostalgic sepia image.

Images - Or Yet Another Application for Arrays
An image can be represented in Python as a 3-dimensional array (H, W, C), with H, W being the height, width of the image respectively, and C referring to the channels. We will represent an image by an array.

To load images into `NumPy` arrays we will use `OpenCV`. `NumPy` is a standard package that can be installed via `conda`. By now everyone should have installed `NumPy`. If it is not automatically installed when installing your anaconda version, it is recommended to install `NumPy` via Conda [6]. `OpenCV` can be installed using `pip`:

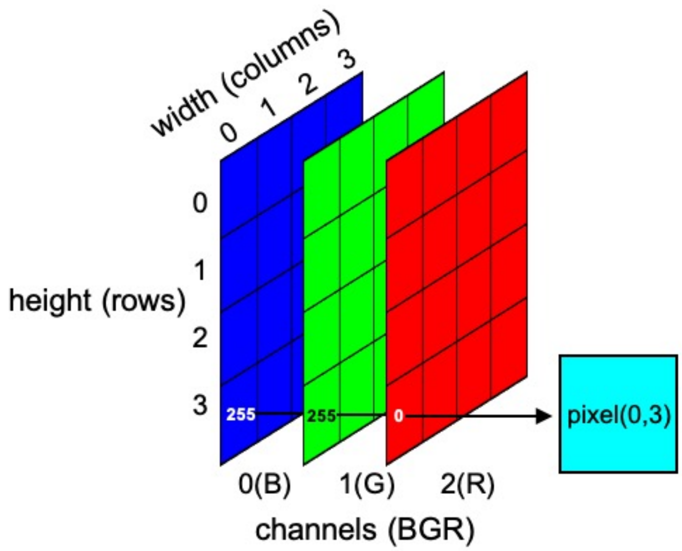
```
pip install opencv-python
```

[6] //numpy.org/install/

After installing these libraries, an image stored in `filename` can be loaded to a `NumPy` array as

```python
import cv2
image = cv2.imread(filename)
```

`image` is then a 3-dimensional `NumPy` array with channels $C = 3$, in the order blue, green and red (BGR)(see graphic below). **Note** that OpenCV uses BGR, while many other image handling libraries use the order red, green and blue (RGB).



Thus, to use `image` with other image libraries (for example for plotting), you need to switch the channel order. This can be done manually on the `NumPy` array, or by the OpenCV call:

```python
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

Grayscale Filter

Task: Write a function `grayscale_filter` that takes an image and returns the gray-scale version of the image. **How to do so?** In the most simplistic approach the conversion to gray-scale takes the weighted sum of the red, green, and blue values, which determines the gray value.

Our approach will take into account that human perception is most sensitive to green and least sensitive to blue. Therefore we apply 0.21, 0.72, and 0.07 as weights for the red, green, and blue channel, respectively. *Note 1:* The weighting has to be applied to each pixel.

To develop your function you can, for example, use the provided image `rain.jpeg`. Display the grayscale image and compare it to the original in order to check that your function is doing what it is supposed to do.

Save the newly created piece of art with the name of the original file and the extension `_grayscale`. For example the grayscale version of `rain.jpeg` would be saved as `rain_grayscale.jpeg`

Important: There are libraries that can apply grayscale for you, but you are not supposed to use these. Please implement the operation manually for this assignment.[7]

Note 2: OpenCV reads images as (unsigned) integers. However, after applying the grayscale function your image likely contains floating point numbers. Thus, for OpenCV to understand your converted image, you have to convert the values back to integers. This can be done as follows

```
grayscale_image = grayscale_image.astype("uint8")
```

Once the grayscale image is converted to the correct type, you can save it to a file using `imwrite`.

```
cv2.imwrite("rain_grayscale.jpeg", dst)
```

Python Implementation (3 points)

Create a Python script implementing the function `python_color2gray` which turns an image into its grayscale new version. *Make sure you read the Notes in the introduction to this assignment.* Make sure to write a *pure Python* implementation (no usage of `NumPy` and vectorization). You are allowed to use `NumPy` **only** for storing the image, if needed. All computations should be done with *pure Python*.

The computation time could grow very large in this task when using images with higher resolution, size or width.

Consider using a small image for testing during development, or resize the provided image using OpenCV.

The following line of code halves the dimensions of the image:

```
image = cv2.resize(image, (0, 0), fx=0.5, fy=0.5)
```

We expect you also to prepare a report containing the dimensions of the image being grayscaled (that is, H, W, C), along with the runtime of the function `python_color2gray.py` in a report `python_report_color2gray.txt`.

The report for the runtime of the picture conversion should contain at least the following information:

```
Timing: python_color2gray
Average runtime running python_color2gray after 3 runs: x.xxxxxx s
Timing performed using: xyz
```

[7] It might actually be fun!

Summary Files to be created in this Subtask

- python_color2gray.py
- python_report_color2gray.txt

NumPy Implementation (3 points)

Make a script similar to the "Python Implementation" python_color2gray.py so that all computationally heavy bits use vector operations. Try to replace your for-loops with NumPy slicing. Compare the runtime on some input of your choosing. How much is gained by switching to NumPy? Your report should contain the dimensions of the image being grayscaled (that is, H, W, C), along with the runtime for each script.

The report for the runtime of the picture conversion should contain at least the following information:

```
Timing: numpy_color2gray
Average runtime running numpy_color2gray after 3 runs:
Average runtime running of numpy_color2gray is x times faster or
    slower than python_color2gray
Timing performed using: xyz
```

You are <u>not</u> supposed to use any library function that applies the grayscale for you. That is, you are not allowed to use for example cv2.cvtColor(), skimage.color.rgb2gray() or PIL.Image.convert(). Instead, convert the function you wrote in python_color2gray.py to a vectorized version.

Summary Files to be created in this Subtask

- numpy_color2gray.py
- numpy_report_color2gray.txt

Numba Implementation (3 points)

Perform your task from "Python Implementation" python_color2gray.py again but this time using Numba to speed things up. Compare the runtime to the initial pure Python as well as to the NumPy implementation. Can you think of any advantages/disadvantages to using Numba instead of NumPy? **Name them in your report.**

Remember that we are, as in the other tasks, expecting a report of the runtime:

```
Average runtime running numpy_color2gray after 3 runs:
Average runtime for running numba_color2gray is x.xxx times faster
    /slower than python_color2gray.py.
Average runtime for running numba_color2gray is x.xxx times faster
    /slower than numpy_color2gray.py
Timing performed using: xyz
```

Summary Files to be created in this Subtask

- numba_color2gray.py
- numba_report_color2gray.txt

Cython Implementation (IN4110 ONLY: 2 points)
Redo your implementation in Cython, and create a report as before. Remember that we are, as in the other tasks, expecting a report of the runtime:

```
Average runtime running numpy_color2gray after 3 runs:
Average runtime for running cython_color2gray is x.xxx times faster/slower than python_color2gray.py.
Average runtime for running cython_color2gray is x.xxx times faster/slower than numpy_color2gray.py
Average runtime for running cython_color2gray is x.xxx times faster/slower than numba_color2gray.py
Timing performed using: xyz
```

Summary Files to be created in this Subtask

- cython_color2gray.py
- cython_report_color2gray.txt

4.2 Sepia Filter - Add Vintage Style to your Images (9 IN3110 + 3 IN4110)

Since we eventually want to turn our instagram filters into a package, we might want to explore more than one function. A sepia filter will add a nice touch to your images.

To display a source image in sepia we need to average the value of all colour channels and replace the resulting value with sepia color. As image editing is a science in itself, smart people came up with the best values for applying weights in order to achieve a nice sepia filter.

Here is the sepia filter matrix in **RGB** order you will be using in this task. You can multiply each color value in the corresponding channel of a pixel with the **RGB** ordered matrix given here:

```

    sepia_matrix = [[ 0.393, 0.769, 0.189],
                  [ 0.349, 0.686, 0.168],
                    [ 0.272, 0.534, 0.131]
```

Each row corresponds to a channel RGB, and the order of the values denote the weights for the colors RGB in each respective channel. *Note 1:* When solving this issue, you might want to think about which order CV2 uses.[8]

Note 2: Because the input image is read as an array of unsigned 8-bit integers (uint8), adding such values will cause an overflow when the sum exceeds

[8] Yes, this is a hint!

255. To combat such overflows, one can for example set the maximum value to 255 for each channel.

Hint: Simply setting pixels to `min(255,pixel_value)` just results in a truncation of the pixel values at 255 instead of keeping the ratios between the pixels in the image at the correct value relative to each other. The alternative would be to scale all pixel values in the image down so that the maximum value is 255. If you want to implement it with the scaling, you are free to do so. However, you do not have to.

You are here as well not supposed to use any library function that applies the sepiafilter for you. As for the grayscale filter, make a

- pure python implementation
- a NumPy implementation
- a Numba implementation
- **for IN4410 ONLY** a Cython implementation (3 points)

for the sepia filter.

Summary Files to be created in this Subtask

- `python_color2sepia.py`
- `python_report_color2sepia.txt`
- `numpy_color2sepia.py`
- `numpy_report_color2sepia.txt`
- `numba_color2sepia.py`
- `numba_report_color2sepia.txt`
- `cython_color2sepia.py`
- `cython_report_color2sepia.txt`

4.3 Develop Your First Package (6 points)

Turn your implementation into a Python package. Create a setup script.[9] Your package should be called `instapy`.

Include a function `grayscale_image(input_filename, output_filename=None)` which returns a numpy (unsigned) integer 3D array of a gray image of `input_filename`. If `output_filename` is supplied, the created image should also be saved to the specified location with the specified name.

[9] `https://www.uio.no/studier/emner/matnat/ifi/IN3110/h21/forelesningsvideoer/2021-09-15-python-2.mp4?vrtx=view-as-webpage`

The function `sepia_image(input_filename, output_filename=None)` should be implemented in the same way as `grayscale_image()`.

Further, make at least two unit tests. Have the tests generate a 3-dimensional numpy array with pixel values randomly chosen between 0 and 255.

One test should check the grayscale_filter functions in **all implementations**.

One test should check the sepia_filter functions in **all implementations**. *Hint:* The test should include checking a random $[i, j]$ index in the sepia image against the expected weighted values.

Use the testing framework `pytest`.

Name of files: `test_instapy.py`, `setup.py`

4.4 User interface (6 points)

Create a folder named `bin` and inside it make a script that implements a user interface for your package. The user interface should use `ArgumentParser` from the library `argparse`. The design of this is up to you, but it should provide instructions by calling it with a `--help` flag, and it should be possible to specify the input and output image filename (for example as passed arguments). It should also be possible to switch between your 3 implementations with a command line argument. Modify your `setup.py` file to install the script. [10].

You should enable a user-friendly implementation that can be called by

```
# option 1
python instapy <arguments>
# or option 2
instapy <arguments>
```

We prepared a list of **expected flags** shown below. You are free to create additional methods that add useful functionality.

These are flags that we expect to be implemented:

```
-h, --help                helpful message showing flags and usage
    of instapy.

-f FILE, --file FILE      The filename of file to apply filter to.

-se, --sepia                Select sepia filter.

-g, --gray                Select gray filter.
-sc SCALE, --scale SCALE        Scale factor to resize image.

-i {python,numba,numpy,cython}, --implement {python,numba,numpy,
    cython}    Choose the implementation.

-o OUT, --out OUT       The output filename.
```

[10]You can read more about how the script keyword, which was shown in the lecture, and console scripts entry point work here `https://bit.ly/393E690` in order to include the commandline tool

Name of file: `instapy.py`

4.5 Stepless Sepia Filter (1 Bonus Point)

Bring your sepia filter to the next level by making it stepless! Allow the user to adapt the level of sepia he or she wants. Implement a stepless sepia filter, where the user can pass in a commandline argument from 0-1 in order to define 0-100% Sepia effect.

4.6 Runtime Tracking Functionality in Instapy (2 Bonus Points)

Implement an additional flag `-r, --runtime` that allows the user to track the average runtime spent on the task with the chosen implementation. The user will receive the average time over 3 runs printed to the terminal, e.g

```python
print(f"Average time over 3 runs: {runtime}")
```

Good job! - You finished more than half of the assignments!