MTH766P: Programming in Python

Please answer all the questions below by writing your own Python code. Using or copy-pasting pre-existing code from any source is not permitted, and will be considered as *plagiarism* and addressed according to College Regulations. You are not allowed to collaborate with other students or ask for help from any other source, including, but not limited to, online forums or ChatGPT. By submitting your Notebook via QMPlus, you confirm that you have followed these rules and that the submitted code is your own work. Upon violation, you will fail this test with zero marks.

General instructions

First, enter your **Full Name** and **Student ID** in the corresponding markdown fields in the section "Student information" below. Then read the exercises and try to work out a solution.

For **Markdown** exercises, **replace all occurrences of** `YOUR ANSWER HERE` **by valid Markdown** to answer the question.

For **coding** exercises, **replace all occurrences of**

```
### YOUR CODE HERE
raise NotImplementedError()
```

by valid Python code to solve the problem. In your solution, you can **only use** built-in Python functions and **functions from modules that you have been explicitly instructed to import**. Importing other modules is not allowed and will result in withdrawal of the corresponding marks. If your code produces an error and you cannot fix it, write a comment to show that you are aware of the problem and indicate possible causes if you can. You can get **partial credit**, so even if you cannot solve an exercise completely, try to answer as much as you can.

Finally, before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart Kernel) and then **run all cells** (in the menubar, select Cell→Run All (Jupyter Notebook Version 6) or Run→Run All Cells (Jupyter Notebook Version 7)). Please **do not change the file name** when you upload your Notebook to QMPlus.

Student information

Full Name:

YOUR ANSWER HERE

Student ID:

YOUR ANSWER HERE

Independent Project Assignment

Important information

Read the following information carefully before turning to the actual exercises. Some aspects of this final project assignment will be handled differently from the in-term exams.

Contribution

This final project assignment is scored on a scale of 100 marks and will contribute **50 % of your final mark** for this module.

Submission details

This project is due by **Friday, 12 January 2024, 17:00 GMT**. Submissions received after this deadline will be treated in accordance with the College Regulations for late submissions: Up to seven days after the deadline, late submissions will incur a penalty of 5 % of the total marks per 24 hours (or a fraction thereof). For example, a submission received on 13/01/2024 at 18:00 GMT (25 hours after the deadline) will result in a deduction of 10 marks. Any submission received more than 168 hours after the deadline will receive zero marks.

The lecturer will be available to answer reasonable questions about this assignment until **Friday, 5 January 2024, 17:00 GMT**. You can contact the lecturer by email (l.dabelow@qmul.ac.uk).

You must use this Jupyter Notebook to answer all exercises. You are not allowed to remove any cells from this notebook. You can add Markdown or code cells as needed.

You must submit this Jupyter Notebook through QMPlus, using the submission system for this assignment. You cannot submit your attempt in any other way. Files sent by other means, including email, will not be considered.

You cannot submit any additional files apart from this Jupyter Notebook. The ZIP file you downloaded contains various auxiliary files that you will need for the exercises. However, there is no need to upload these auxiliary files. The original versions of these files will be merged into your submission automatically.

Coding style

General recommendations

When you are asked to write code, follow the best-practice approach introduced in the lectures. Start by thinking what the code should do and what the steps to achieve this are *without* writing any code at first. It will be a good idea to map out your strategy using appropriate comments. Then fill in the code to implement your strategy.

Comments

Comments are an essential part of programming. Use appropriate comments to explain the reasoning behind your code.

Variables

Use the variable names specified in each exercise when writing your code. For any additional variables you define, choose meaningful, concise names.

Functions

Use the function names specified in each exercise when writing your code. For every function, include a docstring that briefly explains what the function does, what its arguments are, and what value(s) it returns.

It is perfectly fine if you define extra auxiliary functions to carry out a task. This is to say, even if the exercise only asks you to write a function `example_function`, you are free to add other functions like `example_function_aux`, `some_other_function`, ... and use them in your implementation of `example_function`. Choose meaningful names for all auxiliary functions, too.

Assessment criteria

The first thing I will do to evaluate your submission is to clear all outputs and run the entire notebook from the beginning to the end. It is your responsibility to ensure that the code produces **no errors**. Contrary to the in-term assessments, I will not try to fix any errors when assessing your work because you have several weeks to develop this project. Instead, you will receive **zero marks for any exercises where the code cell raises an error**. If you run into an error and do not know how to fix it, comment out the critical line of code and any subsequent ones that depend on it. Add a remark indicating that you are aware of the problem and potential reasons.

If you do not know how to implement a certain operation in Python, add a comment and explain in plain

English (and in sufficient detail) what you would like to do. You may receive partial credit for expedient ideas formulated in this way.

As indicated in the exercise headings below, roughly **20 % of the marks** for each coding exercise will be awarded for **coding style**. This includes the use of appropriate variable names, suitable comments to explain complex sequences of operations, and docstrings for functions. This is to say, even if your code is fully functional and solves the exercise, you may lose marks if the reasoning is not explained properly.

Admitted features and imports

As in the in-term exams, you are free to use any Python concepts or features that are built into the language. Furthermore, you can use the *numpy* and *matplotlib.pyplot* packages. Please run the following code cell to import these two modules. **You must not import any additional modules in any of the code you write.**

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Introduction

In this project, you are going to develop a few tools for basic image manipulation.

A digital image is a two-dimensional array or matrix of *pixels*. Each **pixel** has a *color* and represents a tiny, monochromatic, square patch of the full image. A common way to encode **colors** is to specify them as a mixture of the basic colors *red* (R), *green* (G), and *blue* (B). These are also known as the **channels** and we will work with an image format where each channel for each pixel takes a float value between 0 and 1, known as the **brightness** of the channel. Here 0 means that the corresponding basic color is completely dark, whereas 1 means that it is maximally bright. The following table lists a few examples of common standard colors (in the PDF version, the text color of the bullet points in the "Color" column may not reflect the correct color):

R	G	B	Color
0	0	0	● black
1	0	0	● red
0	1	0	● green
0	0	1	● blue
1	1	0	● yellow
0.5	0	0.5	● purple
0.5	0.5	0.5	● gray
1	1	1	● white

Throughout this project, we will use numpy arrays to process the image data in Python. In our standard format, the images are represented by *three-dimensional* arrays, i.e., arrays taking three indices. The first index corresponds to the row (vertical position) of the pixel, the second index corresponds to its column (horizontal position), and the third index indicates the color channel of the pixel (0 for R, 1 for G, 2 for B). The full image will thus be an array of shape `(height, width, 3)`, where `height` and `width` are the number of pixels in the vertical and horizontal directions, respectively.

For example, assume that we have an image whose pixel in the 4th row and 10th column has RGB color `(0.25, 0.7, 1.0)`. The corresponding row index is `3`, the column index is `9`, and the numpy array will have values

```
image[3, 9, 0] = 0.25 # red channel of pixel (3, 9)
image[3, 9, 1] = 0.7 # green channel of pixel (3, 9)
image[3, 9, 2] = 1.0 # blue channel of pixel (3, 9)
```

to indicate the R, G, and B values of the pixel at position `(3,9)`.

Exercise 1: Loading and saving images [30 marks]

You will know that there are numerous existing formats for storing images in files, such as JPG, PNG, GIF, BMP, ... These formats are all binary and some of them use special compression algorithms to store the image information more efficiently than just providing the RGB values of all pixels. For this exercise, however, we will use our own, very simple, text-based format which does precisely this, it is just a text file with the red, green, and blue values of each pixel.

(a) Channel-wise import [5 marks (incl. 1 mark for coding style)]

In the notebook directory, you will find three files named `queens_building.R.txt`, `queens_building.G.txt`, and `queens_building.B.txt`. These contain the R, G, and B channels, respectively, of an inside view of Queens' Building at Mile End Campus. For example, in the file `queens_building.R.txt`, every number gives the brightness level of the red channel of one pixel. Every line in the file represents a row of pixels, and within each row, the columns are separated by whitespaces. In other words, the jth number in the ith line of the file is the brightness of the red channel of the pixel (i, j). Similarly, `queens_building.G.txt` and `queens_building.B.txt` contain the brightness values of the green and blue channels, respectively.

Note: If you open the files with a text editor, you may not see line breaks depending on your operating system (and your text editor). Unfortunately, Windows uses different conventions for how to represent line breaks than Mac OS X and Linux. However, you don't need to worry too much about this, Python will see the line breaks regardless of what your operating system is. So if you read the file using a suitable Python function, you should definitely find different lines.

Write some Python code that loads the contents of the three files `queens_building.R.txt`, `queens_building.G.txt`, and `queens_building.B.txt` into numpy arrays `image_R`, `image_G`, and `image_B`, respectively. Then combine the data of the R, G, and B channels into a single, three-dimensional numpy array `image` as sketched in the introduction: The first and second indices of the array `image` should refer to the position of the pixel (row and column) and the third index should refer to the channel (`R=0`, `G=1`, `B=2`). For example, the entry `image[5, 8, 1]` should be the brightness of the green channel (third index is `1`) of the pixel in the 6th row (first index is `5`) and 9th column (second index is `8`).

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

(b) Displaying images [3 marks (incl. 0 marks for coding style)]

Display the imported image using the `matplotlib.pyplot.imshow` function. (To check that your import function works correctly, you will find a JPG version of the image, `queens_building.jpg`, in the notebook folder, which you can open with your favorite image viewer.)

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

Note: If you did not manage to import the Queens' building image correctly from the channel data in part (a), please uncomment the second line in the following code cell. This will define an alternative image (the Jupyter logo) which you can use for the following exercises.

In [ ]:
```
# Uncomment the following line of code if you did not manage to import the Queens' Building imag
#image = np.array([[[1.0,1.0,1.0],[1.0,1.0,1.0],[1.0,1.0,1.0],[1.0,1.0,1.0],[1.0,1.0,1.0],[1.0,1
```

(c) Custom-format export [10 marks (incl. 2 marks for coding style)]

Having separate files for the R, G, and B channels is a bit inconvenient. Instead, we would like to store our images in a custom text-based format as follows:

- individual lines correspond to the *rows* of the image;
- *columns* are separated by commas;

- the numerical values of the three channels for each pixel are separated by spaces.

For example, a 3x4 image with red pixels in the first row, (medium) gray pixels in the second row, and purple pixels in the third row will be stored like this:

```
1.0 0.0 0.0,1.0 0.0 0.0,1.0 0.0 0.0,1.0 0.0 0.0
0.5 0.5 0.5,0.5 0.5 0.5,0.5 0.5 0.5,0.5 0.5 0.5
0.5 0.0 0.5,0.5 0.0 0.5,0.5 0.0 0.5,0.5 0.0 0.5
```

Write a function `save_image` that takes a file name and an image (in the form of a numpy array as obtained in Exercise 1b above) and stores the image in the format as defined above. The function should return `True` if the file was written successfully or `False` if an error occurred. Use your function to save the Queens' Building `image` as `queens_building.txt` in the notebook folder.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

(d) Custom-format import [12 marks (incl. 2 marks for coding style)]

Write the corresponding import function `load_image` that takes the name of a file in our custom image format and returns the numpy image array or `None` if an error occurs. Use your function to re-import the image from `queens_building.txt` and assign it to a variable `image2`. Verify that the original `image` and the re-imported `image2` contain the same data by checking that the brightness values of all pixels and channels agree.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

Exercise 2: Image transformations [40 marks]

In this exercise, you will explore a few ways to transform images.

You will still need the previously imported image of Queens' Building. If you did not manage to import this image from the hard drive, continue to use the alternative Jupyter logo image whose definition can be found in the code cell below the paragraph starting with "Note" in Exercise 1b.

(a) Grayscale [5 marks (incl. 1 mark for coding style)]

Write a function `grayscale` that takes an image (as a numpy array as before) and converts it to a *grayscale image*. A grayscale image is an array of shape `(height, width)` whose values encode the *average brightness* of the pixels on a scale from 0 to 1, i.e., the average over the three color channels. For example, assume that the color image has values

```
image[3, 9, 0] = 0.25
image[3, 9, 1] = 0.7
image[3, 9, 2] = 1.0
```

corresponding to the brightnesses of the red, green, and blue channels of the pixel at $(3, 9)$. Then the corresponding array `image_gray` representing the grayscale value should have an entry `image_gray[3, 9] = 0.65` because $\frac{1}{3}(0.25 + 0.7 + 1.0) = 0.65$.

Use your function to obtain a grayscale version of the Queens' Building `image`, assign it to a variable `image_gray`, and display it.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

(b) Color rescaling (1) [10 marks (incl. 2 marks for coding style)]

We can apply functions $f : [0, 1] \to [0, 1]$ to all channels of all pixels to adjust the colors of our image. Consider the mapping

$$f(x) = (1-\alpha)(1-x)^\gamma + \alpha x^\gamma \tag{1}$$

for arbitrary $\alpha \in [0,1]$ and $\gamma > 0$.

Choose four meaningful combinations of α and γ values and verify by means of a suitable plot that this function $f(x)$ indeed maps the interval $[0,1]$ onto itself. Add axes labels and a legend to your plot.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

(c) Color rescaling (2) [5 marks (incl. 1 mark for coding style)]

Write a Python function `rescale_colors` that applies the above rescaling transformation for given α and γ to an image (in the form of a 3D numpy array as in Exercise 1) and returns the transformed image. Display transformed images of the Queens' Building `image` for the choices $\alpha = 0, \gamma = 1$ and $\alpha = 1, \gamma = 2$.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

(d) Color rescaling (3) [5 marks]

Comment on the meaning/role of the parameters α and γ. (You might want to explore further combinations of α and γ to get an idea of what they do.)

Type your answer in the following Markdown cell.

YOUR ANSWER HERE

(e) Blur filter [15 marks (incl. 3 marks for coding style)]

Write a function `blur` that takes an image and returns a blurred version of it. The blur filter should work as follows: The color of each pixel is replaced by a *weighted average* over the colors of the pixel itself and neighboring pixels within a specified *radius* r. Use the Euclidean distance to determine which neighboring pixels should be included. The *relative weight* of a pixel at a distance d from the current pixel should be $\frac{r-d}{r}$. The absolute weight of a pixel is its relative weight divided by the sum of the relative weights of all contributing pixels, i.e., all pixels within the radius r.

For example, if the central pixel is at position $C = (3,7)$, then the distance to a second pixel at position $B = (1,4)$ is

$$d(B,C) = \sqrt{(B_1 - C_1)^2 + (B_2 - C_2)^2} = \sqrt{(1-3)^2 + (4-7)^2} = \sqrt{13} \approx 3.6\,. \tag{2}$$

Hence the pixel at B is within a radius of $r = 4$ from C, but outside a radius of $r = 3$. If the radius is $r = 4$, for example, then the relative weight of the pixel at B in the average to determine the value at position C should be

$$\frac{r-d}{r} = \frac{4-\sqrt{13}}{4} \approx 0.0986\,. \tag{3}$$

Generate blurred versions of the Queens' Building `image` with radii $r = 2$ and $r = 4$ and display them along with the original image in a suitable plot.

Note: The code you will need to write to solve this task might take some time to execute. It takes 25-30 seconds to run my solution on my laptop. If you like, you can test your solution with the smaller `jupyter_logo.txt` image from the notebook folder, which is in our custom text-based format as well (so you should be able to import it using your `load_image` function from Exercise 1d).

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

Exercise 3: Color distributions [30 marks]

In this exercise, you will analyze some statistical properties of images.

(a) Histograms for individual channels [10 marks (incl. 2 marks for coding style)]

Generate and display histograms for the R, G, and B channels over all pixels of the Queens' Building `image`. Add suitable labels for the x and y axes and a title for each plot.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

(b) Dominant colors [10 marks (incl. 2 marks for coding style)]

Write a function `color_frequencies` that returns the relative frequencies of (classes of) colors in an image.

Arguments: Besides the image (as a 3D numpy array as before), the function `color_frequencies` should take one more argument `num_levels` which determines how many different levels of brightness should be distinguished in every channel. For example, if `num_levels = 4`, then you should have 4 different classes or "bins" per color channel, hence a total of $4^3 = 64$ different "fundamental colors classes" that are distiguished.

To avoid ambiguities, first rescale the brightness values of all pixels and channels by multiplying them by 255 and rounding them to the nearest integer. This will result in a numpy array of integer values in the range $0, \ldots, 255$. These are the *byte values* of the channels, and the color class of each pixel should be determined based on these byte values. This means that, for each channel, the full range of 256 byte values should be subdivided into `num_levels` intervals or "bins" comprising an equal number of byte values (plus/minus 1 given the discrete nature of the byte values).

Example: Assume that we have an image with the following absolute frequencies of colors (after rescaling):

color (R, G, B)	number of pixels of that color
(0, 0, 0)	10
(25, 25, 0)	5
(155, 230, 100)	5
(230, 205, 50)	5
(255, 0, 255)	15

Using `num_levels = 2`, we only distinguish two brightness levels per channel: "dark" (D) takes all values in $[0, 128)$ and "light" (L) takes all values in $[128, 256)$. Adding a column that indicates the brightness levels for each channel of each color, we can augment the table as follows:

color (R, G, B)	brightness levels (R, G, B)	number of pixels of that color
(0, 0, 0)	(D, D, D)	10
(25, 25, 0)	(D, D, D)	5
(155, 230, 100)	(L, L, D)	5
(230, 205, 50)	(L, L, D)	5
(255, 0, 255)	(L, D, L)	15

This identifies three different color classes in our image: `(D, D, D)` receives a total of `15` counts from pixels of colors `(0, 0, 0)` and `(25, 25, 0)`; `(L, L, D)` receives a total of `10` counts from pixels of colors `(155, 230, 100)` and `(230, 205, 50)`; and `(L, D, L)` receives a total of `15` counts from just the pixels of color `(255, 0, 255)`.

Return value: The function `color_frequencies` should return two lists of equal length: The first list contains the *mean colors* of the various classes and the second list contains the corresponding *relative frequencies* of colors from each class. Note that the order of the classes does not matter, but the mean colors and frequencies must be in one-to-one correspondence, i.e., the first frequency must correspond to the first

mean color, the second frequency to the second mean color, and so on. The mean color should have brightness values scaled back to $[0, 1]$ and is the average over all colors that would be mapped to the class, not over the colors that can actually be found in the image.

For instance, the mean color of the class `(D, D, D)` in the example from above is `(63.5, 63.5, 63.5)` in the byte scale and thus `(0.25, 0.25, 0.25)` in the float scale (rounded to two decimal places), the mean color of the `(L, L, D)` class is `(191.5, 191.5, 191.5)` in the byte scale and thus `(0.75, 0.75, 0.25)` in the float scale (rounded to two decimal places), and the mean color of the `(L, D, L)` class is `(191.5, 63.5, 191.5` in the byte scale and thus `(0.75, 0.25, 0.75)` in the float scale (rounded to two decimal places). Since the corresponding relative frequencies are $\frac{15}{40} = 0.375$, $\frac{10}{40} = 0.25$, and $\frac{15}{40} = 0.375$ as determined above, the overall return value of `color_frequencies` in this example should be (mean colors rounded to two decimal places)

```
[ [0.25, 0.25, 0.25], [0.75, 0.75, 0.25], [0.75, 0.25, 0.75] ], [0.375, 0.25, 0.375]
```

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

(c) Sorting and visualization [10 marks (incl. 2 marks for coding style)]

Use your function `color_frequencies` to extract the color classes and frequencies of the Queens' Building image using 4 levels per channel. Sort the color classes by their frequency and find a suitable visualization.

Note: If you did not manage to write a working `color_frequencies` function, you can uncomment the second and third lines of the subsequent code cell. They define the mean colors and their frequencies for the `jupyter_logo.txt` image in the notebook folder using 3 levels, which you can use instead for sorting and visualization.

In [ ]:
```
# Uncomment the following two lines of code if you did not find a working color_frequencies func
#mean_colors = [np.array([0.83333333, 0.83333333, 0.83333333]), np.array([0.5, 0.5, 0.5]), np.ar
#frequencies = [0.7706243032330395, 0.05671683389074722, 0.004738015607580824, 0.009336677814938

# YOUR CODE HERE
raise NotImplementedError()
```