Python for Engineers 0509-1820, Semester 1 2019-2020

Homework 9

NUMPY & Image Processing

General instructions

- Read the questions **carefully** and make sure your programs work according to the requirements.
- The homework needs to be done individually!
- Read the submission rules on the course web page. All the questions need to be submitted together in the file ex1_012345678.py attached to the homework, after changing the number 012345678 with your ID number (9 digits, including check digit).
- How to write the solution: in this homework, you need to complete the code in the attached outline file.
- **You are not allowed to change the names of the classes, functions, methods and variables that already appear in the attached outline file.**
- **You are not allowed to erase the instructions that appear in the outline file.**
- Some questions are checked automatically. **You thus need to ensure that the output is exactly fits the requirements (even for spaces).**
- Check your code: in order to ensure correctness of your programs and their robustness in the presence of faulty input, for each question run your program with a variety of different inputs, those that are given as examples in the question and additional ones of your choice (check that the output is correct and that the program does not crash).
- **Unless stated otherwise, you can suppose that the input received by the functions is correct.**
- Final submission date: see course web page.

Question 1

The Ministry of Health wanted to check on the achievements of a new physical training program. For several consecutive months, data on the training candidates (trainees) was collected. The data is collected in *csv* tables (a table contained in a *.csv* file), so that the first column contains the names of the trainees and the first row contains the names of the months. Each row contains data collected at the end of each month. That is, the second column will show the data for each trainee at the end of the first month, the second column at the end of the second month, and so on (see the sample file attached to this homework, *weight_input.csv*, where the weights of the four candidates were measured in kilograms. The first sample month is August).

1. Write a function called *load_training_data* that accepts the path to the *csv* file and returns three objects:
    - *data*: the data of the table, in a Numpy matrix (without the row of months and the column of candidates)
    - *column_names*: list of month names according to the order of the columns (from the first line of the table, without the first character), of type *numpy.array* of strings.
    - *row_names*: list of trainees according to the order of rows (from the first column of the table, without the first character), of type *numpy.array* of strings.

Example of execution on the file attached to this homework:

	August	September	October	November	December	January
Orit	84	81.3	82.8	80.1	77.4	75.2
Miki	79.6	75.2	75	74.3	72.8	71.4
Roni	67.5	66.5	65.3	65.9	65.6	64
Assaf	110.7	108.2	104.1	101	98.3	95.5

```
>>> data, column_names, row_names = load_training_data("weight_input.csv")
>>> print(data)
[[  84.    81.3   82.8   80.1   77.4   75.2]
 [  79.6   75.2   75.    74.3   72.8   71.4]
 [  67.5   66.5   65.3   65.9   65.6   64. ]
 [ 110.7  108.2  104.1  101.    98.3   95.5]]
>>> print(column_names)
['August' 'September' 'October' 'November' 'December' 'January']
>>> print(row_names)
['Orit' 'Miki' 'Roni' 'Assaf']
```

In Sections 2-5 here under:

- **The functions receive as parameters the three output variables of the function of Section 1: *data, row_names, column_names*.**
- **You are not allowed to use loops.**

2. Write a function *get_highest_weight_loss_trainee* that returns the name of the trainee whose weight loss was greatest from the beginning of the program to the end (you may assume there is one). That is, the difference between the start weight and the end weight is greatest. You can use (but it is not required) the *numpy.argmax* function that returns the index where the maximum value is.

Example of execution on the given input table:

```
>>> get_highest_weight_loss_trainee(data, column_names, row_names)
'Assaf'
```

3. Let us define the "monthly difference" as the difference for a particular trainee from one month to the previous month. Write a function *get_diff_data* that returns the difference matrix. The dimensions of this matrix are the same as those of the data matrix, so each column contains the difference between the month of that column and the previous month. The first column is filled with zeros. <u>Note that the input matrix does not change.</u>

   Example of execution on the given input table:

```
>>> get_diff_data(data, column_names, row_names)
array([[ 0. , -2.7,  1.5, -2.7, -2.7, -2.2],
       [ 0. , -4.4, -0.2, -0.7, -1.5, -1.4],
       [ 0. , -1. , -1.2,  0.6, -0.3, -1.6],
       [ 0. , -2.5, -4.1, -3.1, -2.7, -2.8]])
```

4. Write a function *get_highest_loss_month* that returns the name of the month where the sum of monthly difference across all candidates is the largest (i.e., the **loss** of weight was maximum). **You may assume that such a month exists**. Note that the function receives the variables *data, columns_names, row_names* (and not the output matrix from section 3). <u>Use the function from Section 3.</u>

   Example of execution: in the example matrix, in September, the trainers lost 10.6 kg together, an amount greater than in the other months.

```
>>> get_highest_loss_month(data, column_names, row_names)
'September'
```

5. Weight change is known to be proportional to the initial weight. Write a function *get_relative_diff_table* that returns the weight change table. For each participant and each month, the table shows the weight change relative to the previous month, i.e. the "monthly difference" divided by the weight of the previous month. Note that the function receives the values *data, columns_names, row_names* (and not the output matrix from section 3). <u>Use the function from Section 3.</u>

   Example of execution:

```
>>> get_relative_diff_table(data,column_names,row_names)
array([[ 0., -0.03214286,  0.01845018, -0.0326087 , -0.03370787, -0.02842377],
       [ 0., -0.05527638, -0.00265957, -0.00933333, -0.02018843, -0.01923077],
       [ 0., -0.01481481, -0.01804511,  0.00918836, -0.00455235, -0.02439024],
       [ 0., -0.02258356, -0.03789279, -0.02977906, -0.02673267, -0.02848423]])
```

   For example, the figure marked in the output above is the result of the calculation:

$$\frac{november - october}{october} = \frac{74.3 - 75}{75} = -0.0093333$$

Question 2 (this material will be thought in the week of 12/1)

"Geralt of Rivia" roams across the continent in search of monsters that plague the inhabitants of the kingdom. As an emotionless witcher, Geralt chooses his destination based on a cold calculation of cost-benefit.

https://en.wikipedia.org/wiki/The_Witcher_(TV_series)

In this question, we will use a file containing all the destinations and all the monsters that have to be eliminated. The file is in CSV format and contains the following information:

- The first column contains the names of the data columns in the following order:
  - *kingdom*: first column, containing the name of the kingdom in distress
  - *bounty*: second column, containing the reward for eliminating the monster
  - *expenses*: third column, containing the cost of Geralt's journey to the same kingdom
  - *duration*: fourth column, containing the number of days it will take for Geralt to complete the task.
- The rest of the rows contain the relevant information according to the columns as indicated above (see sample table below)
- For your convenience, the following table is included among the homework files, as a file named "missions.csv" (in csv format):

Kingdom	**Bounty**	**Expenses**	**Duration**
Temeria	1000	250	5
Redania	1500	500	3
Kaedwen	500	100	7
Cintra	2500	2000	3

1. Implement the function:
   read_missions_file(file_name)
   - The function receives the name of the data file as defined above (*string*)
   - The function will build a pandas dataframe such that:
     - The table contains 3 columns: Bounty, Expenses, Duration
     - The rows of the table start with the kingdom name, that is the index of the table is according to Kingdom (see example below)
   - In case of an **IO error**, the error should be raised with the caption "An IO error occurred" (see example below).
   - You may assume that if an existing file name is given, the values contained in the file will be valid and the file will contain at least one mission (and one row of column headers).
   - **Hint**: You can (but do not have to) use the *pd.read_csv* command to load the file to the pandas table. Have a look at the function documentation online to pass the appropriate arguments to the function.

2. Implement the function:
   sum_rewards(bounties)
   - The function input is a pandas table that represents the table of missions as returned in Section 1.
   - The function must calculate and return the amount of money that Geralt will receive if he performs all the missions, deducing all expenses related to the mission (see example below).
   - It can be assumed that the rewards of a mission are greater than the expense of the journey to perform the mission.
   - For example, if he performs a mission in the Kingdom of Kaedwen, Geralt will earn 500, but loses 100 on travel expenses. In total, he will earn 400.
   - **Loops may not be used and the function body must be written in one line**

3. Implement the function:
   find_best_kingdom(bounties, kingdoms)
   - The function input is a pandas table that represents the mission table as returned in Section A.
   - The function must find and return the name of the most worthy kingdom. Worthiness of a mission is measured as the mission's reward minus its associated expenses, divided by the number of days it takes to perform the mission.
   - For example: given an array as shown in the table above, the worthiness of the mission to the Kingdom of Temeria is 150 because:
   $$\frac{1000 - 250}{5} = 150$$
   - You may assume that each kingdom appears only once in the file and that there is one and only one worthwhile kingdom.
   - Do not use loops. It is advised, but not mandatory, to write the function body in a single line.

Example of execution:

In the following example, we use the file "missions.csv" attached to the homework and we run the above-defined functions on it. In addition, Section A was run with a file name that does not exist, resulting in an appropriate error being thrown with the required error message (see Section A).

```
In[29]: import pandas as pd
In[30]: file_name = "missions.csv"
In[31]: bounties = read_missions_file(file_name)
In[32]: print(bounties)
         Bounty  Expenses  Duration
Kingdom
Temeria    1000       250         5
Redania    1500       500         3
Kaedwen     500       100         7
Cintra     2500      2000         3
In[33]: read_missions_file("not_real_file.csv")
Traceback (most recent call last):
  File "<ipython-input-28-e708d74fcfa7>", line 5, in read_missions_file
    bounties = pd.read_csv(file_name, index_col="Kingdom")
  File "D:\python\lib\site-packages\pandas\io\parsers.py", line 685, in parser_f
    return _read(filepath_or_buffer, kwds)
  File "D:\python\lib\site-packages\pandas\io\parsers.py", line 457, in _read
    parser = TextFileReader(fp_or_buf, **kwds)
  File "D:\python\lib\site-packages\pandas\io\parsers.py", line 895, in __init__
    self._make_engine(self.engine)
  File "D:\python\lib\site-packages\pandas\io\parsers.py", line 1135, in _make_engine
    self._engine = CParserWrapper(self.f, **self.options)
  File "D:\python\lib\site-packages\pandas\io\parsers.py", line 1917, in __init__
    self._reader = parsers.TextReader(src, **kwds)
  File "pandas/_libs/parsers.pyx", line 382, in pandas._libs.parsers.TextReader.__cinit__
  File "pandas/_libs/parsers.pyx", line 689, in pandas._libs.parsers.TextReader._setup_parser_source
FileNotFoundError: [Errno 2] File b'not_real_file.csv' does not exist: b'not_real_file.csv'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "D:\python\lib\site-packages\IPython\core\interactiveshell.py", line 3319, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-33-e5abd32b322c>", line 1, in <module>
    read_missions_file("not_real_file.csv")
  File "<ipython-input-28-e708d74fcfa7>", line 7, in read_missions_file
    raise IOError("An IO error occurred")
OSError: An IO error occurred
In[34]: print(sum_rewards(bounties))
2650
In[35]: print(find_best_kingdom(bounties))
Redania
```

Question 3 (image processing)

Grayscale images can be represented as a two-dimensional *ndarray*, where each organ is a number in the range 0-255 where 0 represents black and 255 represents white.

Use the *imageio* package to load the images.

1. One of the important measures in information theory is entropy. Entropy is a way to quantify the disorder contained in a word or a picture. Think of an image with just one color, for example black: this picture is very "neat" (note how few words it takes to describe it) and indeed its entropy value is 0. On the other hand, think of a wide-scale image of grayscale: it contains a lot of information and is less "neat". Its entropy is therefore larger.
   In this section, we want to calculate the entropy of a grayscale image and thus quantify the disorder in the image. The formula for calculating the entropy is:
   $$S = \sum_{i=0}^{N} -P_i \cdot \log_2 P_i$$
   where P_i is the probability to get any shade of grey between 0 and 255. In other words, the P_i are the values of the normalized histogram of the picture. *N* is the total number of shades of grey.
   For example, let us compute the entropy of an image containing 4 pixels: 2 white pixels and 2 black pixels:
   $$S = \sum_{i=0}^{N} -P_i \cdot \log_2 P_i = -\frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) - \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) = 1$$

In this specific case, the probability of a black or a white pixel is one half.

Write a function called *def compute_entropy(img)* which receives the name of an image *img* (string) and returns the entropy of the greyscale image.

- You need to discard P_i values that are zero
- You may assume that the input is valid: an image of a given size with shades of grey between 0 and 255.
- You may assume that the image contains more than one shade of grey.
- Hint: use *np.bincount and np.log2*

Example of execution:

```
>>>print(compute_entropy('cameraman.tif'))
>>>7.009716283345514
```

2. When we enlarge an image, we need to find a way to color the new pixels with the help of the old pixels. One of the basic methods for doing this is through the principle of "the nearest neighbor". The pixel in the enlarged image will receive the value of the closest pixel in the original image. The formula for a pixel in the enlarged image will be:

$$Bigpixelvalue[i,j] = smallpixelvalue[floor(i * \frac{ySmallSize}{yBigSize}), floor(j * \frac{xSmallSize}{xBigSize}))]$$

Where *xBigSize* is the size of the *x* axis (number of columns) of the enlarged image, and so on for the other values. The values *i* and *j* are the indexes of the pixels in the enlarged image and $floor(x)$ returns the integer part of x.

Write a function called *nearest_enlarge(img, a)* which:

- Receives the name of the image *img* (*string*) and the enlargement ratio *a*, greater than 1 (*int*)
- Returns an enlarged image (two-dimensional array of type *ndarray*) where the number of pixels of each dimension of the image is multiplied by *a*.
- Hint: use *np.floor*

Example of execution:

```
>>>I=nearest_enlarge('cameraman.tif',2)
>>>plt.figure()
>>>plt.imshow(I,cmap = plt.cm.gray)
>>>plt.show()
```