

Home Assignment 10

numpy, pandas, and 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 `ex10_012345678.py` attached to the homework, after changing the number 012345678 with your ID number (Teudat Zehut if you have one, otherwise it's typically a number beginning with 9).
- How to write the solution: in this homework, you need to complete the code in the attached outline file.
- Do not change the names of the classes, functions, methods, and variables that already appear in the attached outline file.
- Do not erase the instructions that appear in the outline file.
- Some questions are checked automatically. You thus need to ensure that the output exactly matches the requirements (even for spaces).
- Check your code: 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).
- Each question specifies the assumptions you can make about the validity of inputs.
- Submission is due by: see course web page.
- **Make sure your submission contains no reference to local paths or local files.**
- **Make sure your submission contains no debug prints, and no calls to functions between the definitions.**
- **The only place for your test code is at the end of the outline file, inside the if-block**
`if __name__ == "__main__":`

Question 1: numpy

In this question we implement a system of hiding short text messages in images. This cryptographic technique is called [Stenography](#).

As shown in class, grayscale images can be represented as two-dimensional numpy arrays, where each number in the array is an integer between 0 and 255 (uint8 data type) representing the intensity of a single pixel. Recall that the ASCII table maps each character to a small integer; we'll use this to encode our messages within the images.

Given a "carrier" image and a short string, our task is to return a slightly modified image, from which the message can be recovered. To this aim, we'll store within the array representing the image extra information, such as the length of the string and its location within the image.

How to conceal a string within an image? We describe the algorithm step by step.

1. Convert the string to a list of ASCII values using the function `ord`.

Example:

```
>>> print(ord('H'), ord('e'), ord('l'), ord('l'), ord('o'))  
72 101 108 108 111
```

The function `np_array_to_ascii` and its inverse `ascii_to_np_array` are given to you in the outline file. These convert a numpy array of dtype `uint8` to a string and vice versa.

2. Next, we find the best location (row and column) in the image for hiding the message. A location is good if the sequence of numbers beginning at that location are most similar to the numbers we want to hide, so the error introduced by changing the original number to the numbers encoding the string is minimized.
3. The modified image is this produced by:
 - a. Creating a copy of the original image;
 - b. Finding the best location to hide it;
 - c. Overwrite the numbers in that location by the numbers encoding the message;
 - d. Savings extra information in constant locations, according to which the message can be retrieved:
 - Pixel [0, 0] = row index
 - Pixel [0, 1] = column index
 - Pixel [0, 2] = message length

Comments:

- The carrier image is not necessarily square.
- We may assume the width of the image (= number of columns) is greater than the length of the message.
- We may assume the message is not empty.
- Each integer in the array is of dtype `uint8`, and can represent values between 0-255, so each of the two indices and the message length cannot exceed 255.

Step A

Implement the function `arr_dist(a1, a2)`, which gets two arrays of the same length, and computes the distance between them, defined as the sum of absolute values of differences between respective entries. The function should be implemented in a single line of code, with no use of iteration (no for loops, no list comprehension, etc.)

Example:

The distance between the strings “Hello” and “Halmo” is 5:

H (72)	e (101)	l (108)	l (108)	o (111)
H (72)	a (97)	l (108)	m (109)	o (111)
72-72	101-97	108-108	108-109	111-111
0	4	0	1	0

```
>>> a1 = ascii_to_np_array('Hello')
>>> print(a1)
[111 108 108 101 72]
>>> a2 = ascii_to_np_array('Halmo')
>>> print(arr_dist(a1, a2))
5
```

Step B

Implement the function `find_best_place(im, np_msg)`, which gets an image and the array corresponding to a message, and returns, as a tuple of row and column, the location in the image at which the most similar sequence of numbers (of the same length as the message) to `np_msg` begins. Similarity is measured with respect to the metric `arr_dist` defined above. Loops are permitted.

Example:

```
>>> im1 = imageio.imread('parrot.png')
>>> np_msg = ascii_to_np_array('Hello')
>>> idx = find_best_place(im1, np_msg)
>>> print(idx, type(idx))
(110, 121) <class 'tuple'>
```

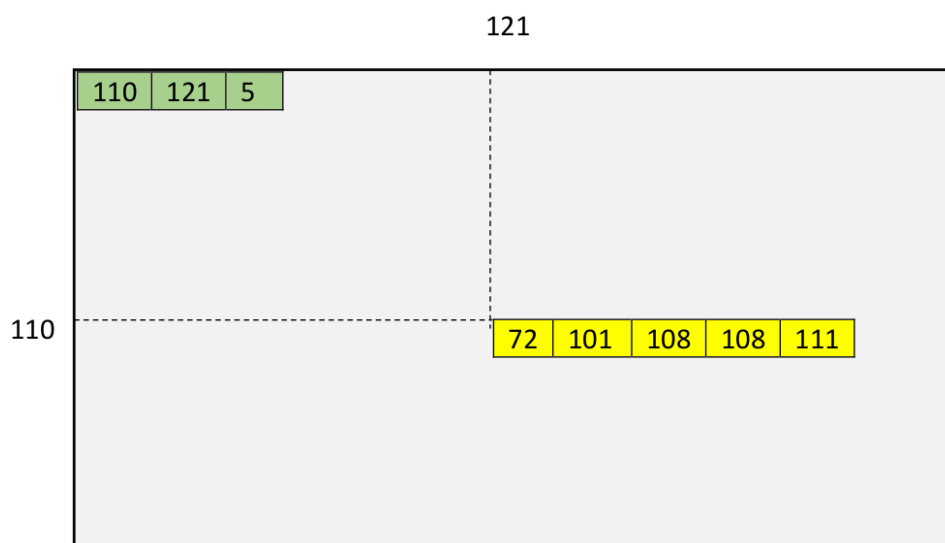
Step C

Implement the function `create_image_with_msg(im, im_idx, np_msg)`, which gets an image, a tuple of indices, and a message-array, and returns a new image in which the message was embedded.

The original image should be copied (see, e.g., the method `copy` of a numpy array), the message-array should be copied to the location specified by the indices `im_idx`, and the first three pixels in the top row should be set to the row index, column index, and message length, respectively. As in Step A, no explicit iteration is allowed in your code.

Example:

For the example from Step B, the message would be embedded like this:



The yellow pixels encode the message itself, and the green pixels encode the extra information. All other pixels are left unchanged.

Step D

Implement the function `put_message(im, msg)`, which gets a carrier image and a string, and returns a new image with the message hidden in it.

The function needs to convert the string to an array, find the best location for it (step B), embed the message there (step C) and return the image. Do not use iteration.

Step E

Implement the function `get_message(im)`, which gets an image and returns the message hidden in it (assume some message was hidden in `im` using `put_message` from step D).

The function needs to use the extra information from the first 3 pixels to retrieve the message-array and convert it back to a string (via `np_array_to_string`). Do not use iteration.

Question 2: pandas

Geralt of Rivia travels across the entire continent, slaying monsters which prey on the poor citizens of the various kingdoms. As a handsome, yet heartless, [Witcher](#), he picks his next quests using a cold-hearted cost vs. benefit calculation.

In this question we make use of a data file containing information about available missions. The file is given in CSV format, and is structured as follows:

- The first row consists of the column headers, in the following order:
 - o Kingdom: name of a kingdom requiring assistance from Geralt;
 - o Bounty: payment for getting rid of that monster;
 - o Expenses: cost of travel to that kingdom;
 - o Duration: number of days required to handle this mission end-to-end.
- Each subsequent row consists of values describing a single quest (see example below).

Example: (available to you in the file `missions.csv`)

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

Part A

Implement the function `read_missions_file(file_name)`, which gets a string specifying a CSV file of the format described above. The function returns a pandas DataFrame with 3 columns: Bounty, Expenses, and Duration. The kingdom names given in the Kingdom column will be used as row indices.

- In case of an I/O error, the function will catch it and raise an `IOError` with the message **“An IO error occurred”** (see example in the next page).
- You may assume that if the file is readable, it has the correct format and contains at least one mission (besides the column headers).
- Hint: read the documentation of `pd.read_csv` to see what optional argument may assist you in loading the data as requested.

Part B

Implement the function `add_daily_gain_col(bounties)`, which gets the DataFrame loaded by the function from part A. No value is returned, but the DataFrame is modified to include a new column named **“Daily gain”**. This value is the net gain per day, i.e., the difference between bounty payment and expenses, divided by the number of days required.

For instance, the daily gain for helping Temeria is $\frac{1000-250}{5} = 150$ (see example above).

In each of the following parts, the input is a DataFrame returned by `read_missions_file`.

Part C

Implement the function `sum_rewards(bounties)`, which returns the total net gain (total payment minus all expenses) for Geralt, assuming he helps everybody.

The body of the function has to consist of a single line of code. Do not use iteration.

- You may assume no mission has a negative gain. For instance, the gain for helping Kaedwen is $500 - 100 = 400$.

Part D

Implement the function `find_best_kingdom(bounties)`, which finds the most lucrative mission and returns the name of the kingdom Geralt should help. Geralt evaluates missions according to daily gain, as defined in part B.

- You may assume each kingdom name appears in the `bounties` DataFrame only once, and that the most lucrative one is unique.
- Besides the ones provided by `read_missions_file`, do not assume `bounties` has any other columns. In particular, call `add_daily_gain_col` if you need that column.

Do not use iteration. Try to see if you can write the function using a single-line body (not counting the call to `add_daily_gain_col`, should you use it).

Part E

Geralt is tired of packing different amount of underwear per mission, and decided to only take missions of one particular duration (so he could prepare his travel bag once, for that amount of days).

To help him decide what duration is most lucrative, he asked you (and we convey his humble request) to implement the function `find_best_duration(bounties)`, which returns a duration (integer) maximizing the average daily gain when the average is taken over missions of the chosen duration.

- Besides the ones provided by `read_missions_file`, do not assume `bounties` has any other columns. In particular, call `add_daily_gain_col` if you need that column.
- Hint: use `groupby`.

Example 1: missions.csv (same as above)

(part A)

```
>>> filename = "missions.csv"
>>> df = read_missions_file(filename)
>>> print(df)
```

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

(part B)

```
>>> add_daily_gain_col(df)
>>> print(df)
```

	Bounty	Expenses	Duration	daily gain
Kingdom				
Temeria	1000	250	5	150.000000
Redania	1500	500	3	333.333333
Kaedwen	500	100	7	57.142857
Cintra	2500	2000	3	166.666667

(part C)

```
>>> sum_rewards(df)
2650
```

(part D)

```
>>> print(find_best_kingdom(df))
Redania
```

(part E)

```
>>> print(find_best_duration(df))
3
```

Explanation: the average daily gain of the two missions of duration 3 is 250; for each of the other two durations, there is but a single mission of that duration, so the average daily gain is just the duration of that mission.

Now, the maximum among 250, 150 and 57.142857 is 250.

Example 2: bad input (non-existing file)

```
>>> read_missions_file("notAFile.csv")
Traceback (most recent call last):
  File "C:\Users\LENOVO\Desktop\python2021\ex10\EX_10_sol.py", line 67, in read_missions_file
    [redacted]
  File "C:\Users\LENOVO\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 686, in read_csv
    return _read(filepath_or_buffer, kwds)
  File "C:\Users\LENOVO\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 452, in _read
    parser = TextFileReader(fp_or_buf, **kwds)
  File "C:\Users\LENOVO\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 936, in __init__
    self._make_engine(self.engine)
  File "C:\Users\LENOVO\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 1168, in _make_engine
    self._engine = CParserWrapper(self.f, **self.options)
  File "C:\Users\LENOVO\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 1998, 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 674, in pandas._libs.parsers.TextReader._setup_parser_source
FileNotFoundError: [Errno 2] No such file or directory: 'notAFile.csv'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    read_missions_file("notAFile.csv")
  File "C:\Users\LENOVO\Desktop\python2021\ex10\EX_10_sol.py", line 69, in read_missions_file
    raise IOError("An IO error occurred")
OSError: An IO error occurred
```

Good luck!