Python & ML - Module 03

Numpy

Summary: Today you will learn how to use the Python library that will allow you to manipulate multidimensional arrays (vectors) and perform complex mathematical operations on them.

Chapter I

Common Instructions

- The version of Python recommended to use is 3.7, you can check the version of Python with the following command: `python -V`
- The norm: during this piscine, it is recommended to follow the PEP 8 standards, though it is not mandatory. You can install pycodestyle which is a tool to check your Python code.
- The function `eval` is never allowed.
- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- If you are a student from 42, you can access our Discord server on 42 student's associations portal and ask your questions to your peers in the dedicated Bootcamp channel.
- You can also ask questions in the `#bootcamps` channel on Slack at 42AI or 42born2code.
- If you find any issue or mistakes in the subject please create an issue on 42AI repository on Github.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be run after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Contents

I	**Common Instructions**	**1**
II	**Exercise 00**	**3**
III	**Exercise 01**	**6**
IV	**Exercise 02**	**9**
V	**Exercise 03**	**12**
VI	**Exercise 04**	**17**

Chapter II

Exercise 00

	Exercise : 00
NumPyCreator	
Turn-in directory : $ex00/$	
Files to turn in : `NumPyCreator.py`	
Forbidden functions : `None`	

Objective

Introduction to the Numpy library.

Instructions

Write a class named `NumPyCreator`, that implements all of the following methods.

Each method receives as an argument a different type of data structure and transforms it into a Numpy array:

- `from_list(self, lst)`: takes a list or nested lists and returns its corresponding Numpy array.
- `from_tuple(self, tpl)`: takes a tuple or nested tuples and returns its corresponding Numpy array.
- `from_iterable(self, itr)`: takes an iterable and returns an array which contains all of its elements.
- `from_shape(self, shape, value)`: returns an array filled with the same value. The first argument is a tuple which specifies the shape of the array, and the second argument specifies the value of the elements. This value must be 0 by default.
- `random(self, shape)`: returns an array filled with random values. It takes as an argument a tuple which specifies the shape of the array.

- `identity(self, n)`: returns an array representing the identity matrix of size n.

BONUS: Add to these methods an optional argument which specifies the datatype (dtype) of the array (e.g. to represent its elements as integers, floats, ...)

`Each of these methods can be implemented in one line. You only need to find the right Numpy functions.`

Examples

```
from NumpyCreator import NumpyCreator
npc = NumpyCreator()

npc.from_list([[1,2,3],[6,3,4]])
# Output :
array([[1, 2, 3],
       [6, 3, 4]])


npc.from_list([[1,2,3],[6,4]])
# Output :
None


npc.from_list([[1,2,3],['a','b','c'],[6,4,7]])
# Output :
array([['1','2','3'],
       ['a','b','c'],
       ['6','4','7'], dtype='<U21'])


npc.from_list(((1,2),(3,4)))
# Output :
None


npc.from_tuple(("a", "b", "c"))
# Output :
array(['a', 'b', 'c'])


npc.from_tuple(["a", "b", "c"])
# Output :
None


npc.from_iterable(range(5))
# Output :
array([0, 1, 2, 3, 4])


shape=(3,5)
npc.from_shape(shape)
# Output :
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])


npc.random(shape)
# Output :
array([[0.57055863, 0.23519999, 0.56209311, 0.79231567, 0.213768 ],
      [0.39608366, 0.18632147, 0.80054602, 0.44905766, 0.81313615],
      [0.79585328, 0.00660962, 0.92910958, 0.9905421 , 0.05244791]])


npc.identity(4)
# Output :
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Chapter III

Exercise 01

	Exercise : 01
ImageProcessor	
Turn-in directory : $ex01/$	
Files to turn in : `ImageProcessor.py`	
Forbidden functions : `None`	

Objective

Basic manipulation of image via the Matplotlib library.

Instructions

Build a tool that will be helpful to load and display images in the upcoming exercises.

Write a class named `ImageProcessor` that implements the following methods:

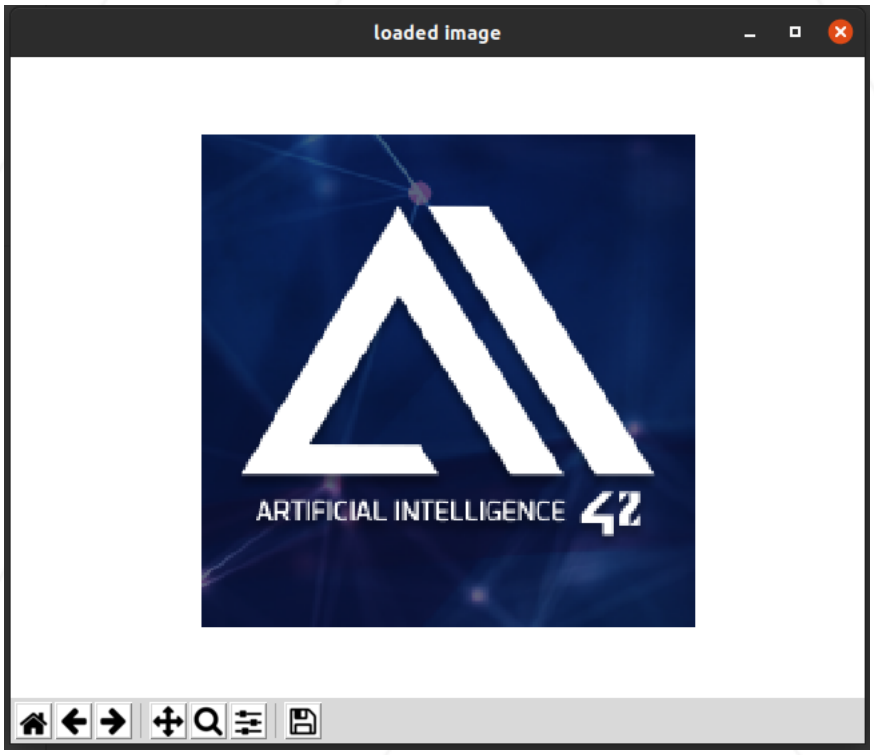
- `load(path)`: opens the PNG file specified by the `path` argument and returns an array with the RGB values of the pixels in the image. It must display a message specifying the dimensions of the image (e.g. 340 x 500).
- `display(array)`: takes a numpy array as an argument and displays the corresponding RGB image.

You must handle these errors: if the file passed as argument does not exist or if it can't be read as an image, with an appropriate message of your choice.

`You can use the library of your choice for this exercise, but converting the image to a numpy array is mandatory. The goal of this exercise is to dispense with the technicality of loading and displaying images, so that you can focus on array manipulations in the upcoming exercises.`

Examples

```
from ImageProcessor import ImageProcessor
imp = ImageProcessor()
arr = imp.load("non_existing_file.png")
# Output :
Exception: FileNotFoundError -- strerror: No such file or directory


print(arr)
# Output :
None


arr = imp.load("empty_file.png")
# Output :
Exception: OSError -- strerror: None


print(arr)
# Output :
None


arr = imp.load("../resources/42AI.png")
# Output :
Loading image of dimensions 200 x 200


arr
# Output :
array([[[0.03529412, 0.12156863, 0.3137255 ],
        [0.03921569, 0.1254902 , 0.31764707],
        [0.04313726, 0.12941177, 0.3254902 ],
        ...,
        [0.02745098, 0.07450981, 0.22745098],
        [0.02745098, 0.07450981, 0.22745098],
        [0.02352941, 0.07058824, 0.22352941]],

       [[0.03921569, 0.11764706, 0.30588236],
        [0.03529412, 0.11764706, 0.30980393],
        [0.03921569, 0.12156863, 0.30980393],
        ...,
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098]],

       [[0.03137255, 0.10980392, 0.2901961 ],
        [0.03137255, 0.11372549, 0.29803923],
        [0.03529412, 0.11764706, 0.30588236],
        ...,
        [0.02745098, 0.07450981, 0.23137255],
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098]],

       ...,

       [[0.03137255, 0.07450981, 0.21960784],
        [0.03137255, 0.07058824, 0.21568628],
        [0.03137255, 0.07058824, 0.21960784],
        ...,
        [0.03921569, 0.10980392, 0.2784314 ],
        [0.03921569, 0.10980392, 0.27450982],
        [0.03921569, 0.10980392, 0.27450982]],

       [[0.03137255, 0.07058824, 0.21960784],
        [0.03137255, 0.07058824, 0.21568628],
        [0.03137255, 0.07058824, 0.21568628],
        ...,
        [0.03921569, 0.10588235, 0.27058825],
        [0.03921569, 0.10588235, 0.27058825],
        [0.03921569, 0.10588235, 0.27058825]],

       [[0.03137255, 0.07058824, 0.21960784],
        [0.03137255, 0.07058824, 0.21176471],
        [0.03137255, 0.07058824, 0.21568628],
        ...,
        [0.03921569, 0.10588235, 0.26666668],
        [0.03921569, 0.10588235, 0.26666668],
        [0.03921569, 0.10588235, 0.26666668]]], dtype=float32)
```



The image must be displayed in a separate window when running in the console.

Chapter IV

Exercise 02

	Exercise : 02
ScrapBooker	
Turn-in directory : $ex02/$	
Files to turn in : `ScrapBooker.py`	
Forbidden functions : `None`	

Objective

Manipulation and initiation to slicing on numpy arrays.

Instructions

Implement a class named `ScrapBooker` with the following methods:

- `crop`,
- `thin`,
- `juxtapose`,
- `mosaic`.

```
# within the class
def crop(self, array, dim, position=(0,0)):
    """
    Crops the image as a rectangle via dim arguments (being the new height
    and width of the image) from the coordinates given by position arguments.

    Args:
    -----
      array:    numpy.ndarray
      dim:      tuple of 2 integers.
      position: tuple of 2 integers.

    Returns:
    -------
      new_arr:  the cropped numpy.ndarray.
      None:     (if the combination of parameters is not possible).

    Raises:
    ------
      This function should not raise any Exception.
    """
    ... your code ...

def thin(self, array, n, axis):
    """
    Deletes every n-th line pixels along the specified axis (0: vertical, 1: horizontal)

    Args:
    -----
      array:   numpy.ndarray.
      n:       non null positive integer lower than the number of row/column of the array
               (depending of axis value).
      axis:    positive non null integer.

    Returns:
    -------
      new_arr:  thined numpy.ndarray.
      None:     (if the combination of parameters is not possible).

    Raises:
    ------
      This function should not raise any Exception.
    """
    ... your code ...

def juxtapose(self, array, n, axis):
    """
    Juxtaposes n copies of the image along the specified axis.

    Args:
    -----
      array:    numpy.ndarray.
      n:        positive non null integer.
      axis:     integer of value 0 or 1.

    Returns:
    -------
      new_arr:  juxtaposed numpy.ndarray.
      None:     (if the combination of parameters is not possible).

    Raises:
    -------
      This function should not raise any Exception.
    """
    ... your code ...

def mosaic(self, array, dim):
    """
    Makes a grid with multiple copies of the array. The dim argument specifies
    the number of repetition along each dimensions.

    Args:
    -----
      array:    numpy.ndarray.
      dim:      tuple of 2 integers.

    Return:
    -------
      new_arr:  mosaic numpy.ndarray.
      None      (combinaison of parameters not compatible).
```

In this exercise, when specifying positions or dimensions, we will assume that the first coordinate is counted along the vertical axis starting from the top, and that the second coordinate is counted along the horizontal axis starting from the left. Indexing starts from 0.

e.g.:

(1,3)

.....

...x.

.....

Examples

```
import numpy as np
from ScrapBooker import ScrapBooker

spb = ScrapBooker()
arr1 = np.arange(0,25).reshape(5,5)
spb.crop(arr1, (3,1),(1,0))
#Output :
array([[ 5],
       [10],
       [15]])


arr2 = np.array("A B C D E F G H I".split() * 6).reshape(-1,9)
spb.thin(arr2,3,0)
#Output :
array([['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K'],
       ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K'],
       ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K'],
       ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K'],
       ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K'],
       ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K']], dtype='<U1')


arr3 = np.array([[1, 2, 3],[1, 2, 3],[1, 2, 3]])
spb.juxtapose(arr3, 3, 1)
#Output :
array([[1, 2, 3, 1, 2, 3, 1, 2, 3],
       [1, 2, 3, 1, 2, 3, 1, 2, 3],
       [1, 2, 3, 1, 2, 3, 1, 2, 3]])
```

Chapter V

Exercise 03

	Exercise : 03
ColorFilter	
Turn-in directory : $ex03/$	
Files to turn in : `ColorFilter.py`	
Forbidden functions : `None`	

Objective

Manipulation of loaded image via numpy arrays, broadcasting.

Instructions

You have to develop a tool that can apply a variety of color filters on images.

For this exercise, the authorized functions and operators are specified for each methods.

You are not allowed to use anything else.

Write a class named `ColorFilter` with 6 methods with the following exact signatures:

```
def invert(self, array):
    """
    Inverts the color of the image received as a numpy array.
    Args:
    -----
        array: numpy.ndarray corresponding to the image.
    Return:
    -------
        array: numpy.ndarray corresponding to the transformed image.
        None: otherwise.
    Raises:
    -------
        This function should not raise any Exception.
    """
```

```
def to_blue(self, array):
    """
    Applies a blue filter to the image received as a numpy array.
    Args:
    -----
        array: numpy.ndarray corresponding to the image.
    Return:
    -------
        array: numpy.ndarray corresponding to the transformed image.
        None: otherwise.
    Raises:
    -------
        This function should not raise any Exception.
    """
```

```
def to_green(self, array):
    """
    Applies a green filter to the image received as a numpy array.
    Args:
    -----
        array: numpy.ndarray corresponding to the image.
    Return:
    -------
        array: numpy.ndarray corresponding to the transformed image.
        None: otherwise.
    Raises:
    -------
        This function should not raise any Exception.
    """
```

```
def to_red(self, array):
    """
    Applies a red filter to the image received as a numpy array.
    Args:
    -----
        array: numpy.ndarray corresponding to the image.
    Return:
    -------
        array: numpy.ndarray corresponding to the transformed image.
        None: otherwise.
    Raises:
    -------
        This function should not raise any Exception.
    """
```

```
def to_celluloid(self, array):
    """
    Applies a celluloid filter to the image received as a numpy array.
    Celluloid filter must display at least four thresholds of shades.
    Be careful! You are not asked to apply black contour on the object,
    you only have to work on the shades of your images.
    Remarks:
        celluloid filter is also known as cel-shading or toon-shading.
    Args:
    -----
        array: numpy.ndarray corresponding to the image.
    Return:
    -------
        array: numpy.ndarray corresponding to the transformed image.
        None: otherwise.
    Raises:
    -------
        This function should not raise any Exception.
    """
```

```
def to_grayscale(self, array, filter, **kwargs):
    """
    Applies a grayscale filter to the image received as a numpy array.
    For filter = 'mean'/'m': performs the mean of RBG channels.
    For filter = 'weight'/'w': performs a weighted mean of RBG channels.
    Args:
    -----
        array: numpy.ndarray corresponding to the image.
        filter: string with accepted values in ['m','mean','w','weight']
        weights: [kwargs] 3 floats where the sum equals to 1,
                  corresponding to the weights of each RBG channels.
                  Expecting keys: 'r_weight', 'g_weight' and 'b_weight'.
    -------
        array: numpy.ndarray corresponding to the transformed image.
        None: otherwise.
    Raises:
    -------
        This function should not raise any Exception.
    """
```

You have some restrictions on the authorized methods and operators for each filter method in class `ColorFilter`:

- `invert`:
  - Authorized functions: `.copy`.
  - Authorized operators: `+`,`-`,`=`.
- `to_blue`:
  - Authorized functions: `.copy`, `.zeros`,`.shape`,`.dstack`.
  - Authorized operators: `=`.
- `to_green`:
  - Authorized functions: `.copy`.
  - Authorized operators: `*`, `=`.
- `to_red`:
  - Authorized functions: `.copy`, `.to_green`,`.to_blue`.
  - Authorized operators: `-`,`+`, `=`.
- `to_celluloid(array)`:
  - Authorized functions: `.copy`, `.arange`,`.linspace`, `.min`, `.max`.
  - Authorized operators: `=`, `<=`, `>`, `&` (or `and`).
- `to_grayscale`:
  - Authorized functions: `.sum`,`.shape`,`.reshape`,`.broadcast_to`,`.as_type`.
  - Authorized operators: `*`,`/`, `=`.

Examples

```
from ImageProcessor import ImageProcessor
imp = ImageProcessor()
arr = imp.load("assets/42AI.png")
# Output :
Loading image of dimensions 200 x 200


from ColorFilter import ColorFilter
cf = ColorFilter()
cf.invert(arr)

cf.to_green(arr)

cf.to_red(arr)

cf.to_blue(arr)

cf.to_celluloid(arr)

cf.to_grayscale(arr, 'm')

cf.to_grayscale(arr, 'weight', r_weight=0.2, 'g_weight'=0.3, 'b_weight'=0.5)
```

The first image is a stylization of Elon Musk that has been generated using a style transfer algorithm implemented in our lab. You can see the code in 42AI repository StyleTransferMirror

Figure V.1: Elon Musk

Figure V.2: Inverted filter

Figure V.3: Blue filter

Figure V.4: Green filter

Figure V.5: Red filter

Figure V.6: Celluloid filter

Chapter VI

Exercise 04

	Exercise : 04
K-means Clustering	
Turn-in directory : *ex*04/	
Files to turn in : `Kmeans.py`	
Forbidden functions : `Any functions allowing you to perform K-Means`	

ALERT! DATA CORRUPTED

Objective

Implementation of a basic K-means algorithm.

Instructions

The solar system census dataset is corrupted! The citizens' homelands are missing!

You must implement the K-means clustering algorithm in order to recover the citizens' origins.

You can find good explanations on how K-means is working here: Possibly the simplest way to explain K-Means algorithm

The missing part is how to compute the distance between 2 data points (cluster centroid or a row in the data).

In our case, the data we have to process is composed of 3 values (height, weight and bone_density).

Thus, each data point is a vector of 3 values.

Now that we have mathematically defined our data points (vector of 3 values), it is very easy to compute the distance between two points using vector properties.

You can use L1 distance, L2 distance, cosine similarity, and so forth...

Choosing the distance to use is called hyperparameter tuning.

I would suggest you to try with the easiest setting (L1 distance) first.

What you will notice is that the final result of the "training"/"fitting" will depend a lot on the random initialization.

Commonly, in machine-learning libraries, K-means is run multiple times (with different random initializations) and the best result is saved.

NB: To implement the fit function, keep in mind that a centroid can be considered as the gravity center of a set of points.

Your program `Kmeans.py` takes 3 parameters: `filepath`, `max_iter` and `ncentroid`:

```
python Kmeans.py filepath='../ressources/solar_system_census.csv' ncentroid=4 max_iter=30
```

Your program is expected to:

- parse its arguments,
- read the dataset,
- fit the dataset,
- display the coordinates of the different centroids and the associated region (for the case `ncentroid=4`),
- display the number of individuals associated to each centroid,
- (Optional) display the results on 3 differents plots, corresponding to 3 combinations of 2 parameters, using different colors to distinguish between Venus,Earth, Mars and Belt asteroids citizens.

Create the class `KmeansClustering` with the following methods:

```python
class KmeansClustering:
    def __init__(self, max_iter=20, ncentroid=5):
        self.ncentroid = ncentroid # number of centroids
        self.max_iter = max_iter # number of max iterations to update the centroids
        self.centroids = [] # values of the centroids

    def fit(self, X):
        """
        Run the K-means clustering algorithm.
        For the location of the initial centroids, randomly pick n centroids from the dataset.
        Args:
        -----
          X: has to be an numpy.ndarray, a matrice of dimension m * n.
        Return:
        -------
          None.
        Raises:
        -------
          This function should not raise any Exception.
        """
        ... your code ...

    def predict(self, X):
        """
        Predict from wich cluster each datapoint belongs to.
        Args:
        -----
          X: has to be an numpy.ndarray, a matrice of dimension m * n.
        Return:
        -------
          the prediction has a numpy.ndarray, a vector of dimension m * 1.
        Raises:
        -------
          This function should not raise any Exception.
        """
        ... your code ...
```

Dataset

The dataset, named **solar_system_census** can be found in the resources folder.

It is a part of the solar system census dataset, and contains biometric data such as the height, weight, and bone density of solar system citizens.

Solar citizens come from four registered areas:

- The flying cities of Venus,
- United Nations of Earth,
- Mars Republic,
- Asteroids' Belt colonies.

Unfortunately the data about the planets of origin was lost... Use your K-means algorithm to recover it! Once your clusters are found, try to find matches between clusters and the citizens' homelands.

- People are slender on Venus than on Earth.
- People of the Martian Republic are taller than on Earth.
- Citizens of the Belt are the tallest of the solar system and have the lowest bone density due to the lack of gravity.

Examples

Here is an example of the K-means algorithm in action:
https://i.ibb.co/bKFVVx2/ezgif-com-gif-maker.gif

Contact

You can contact 42AI association by email: contact@42ai.fr

If you are a student from 42, you can access our Discord server on 42 student's associations portal and ask your questions to your peers in the dedicated Bootcamp channel.

Find all the relevant and up-to-date information about 42AI on our Website !

Thank you for attending this Python Bootcamp module03 !

Acknowledgements

The modules Python & ML is the result of a collective work, we would like to thank:

- Maxime Choulika (cmaxime),
- Pierre Peigné (ppeigne, pierre@42ai.fr),
- Matthieu David (mdavid, matthieu@42ai.fr),
- Quentin Feuillade–Montixi (qfeuilla, quentin@42ai.fr)

who supervised the creation, the enhancement and this present transcription.

- Louis Develle (ldevelle, louis@42ai.fr)
- Augustin Lopez (aulopez)
- Luc Lenotre (llenotre)
- Owen Roberts (oroberts)
- Thomas Flahault (thflahau)
- Amric Trudel (amric@42ai.fr)
- Baptiste Lefeuvre (blefeuvr@student.42.fr)
- Mathilde Boivin (mboivin@student.42.fr)
- Tristan Duquesne (tduquesn@student.42.fr)

for your investment for the creation and development of these modules.

- Barthélémy Leveque (bleveque@student.42.fr)
- Remy Oster (roster@student.42.fr)
- Quentin Bragard (qbragard@student.42.fr)
- Marie Dufourq (madufour@student.42.fr)
- Adrien Vardon (advardon@student.42.fr)

who betatest the first version of the modules of Machine Learning.