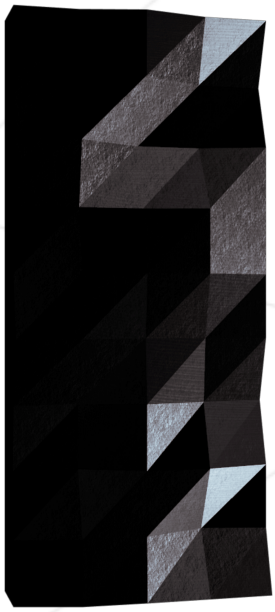


Training Piscine Python for datascience - 1

Array

Summary: Today, you will discover arrays, their manipulations, and work on images.

Version: 1.1

Contents

Chapter I

General rules

- You have to render your modules from a computer in the cluster either using a virtual machine:
    - You can choose the operating system to use for your virtual machine
    - Your virtual machine must have all the necessary software to realize your project. This software must be configured and installed.
- Or you can use the computer directly in case the tools are available.
    - Make sure you have the space on your session to install what you need for all the modules (use the goinfre if your campus has it)
    - You must have everything installed before the evaluations
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- 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 done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.
- You must use the Python 3.10 version
- Your lib imports must be explicit, for example you must "import numpy as np". Importing "from pandas import *" is not allowed, and you will get 0 on the exercise.
- There is no global variable.
- By Odin, by Thor ! Use your brain !!!

Chapter II

Specific instructions of the day

- No code in the global scope. Use functions!
- Each program must have its main and not be a simple script:

```
def main():
  # your tests and your error handling

if __name__ == "__main__":
  main()
```

- Any exception not caught will invalidate the exercices, even in the event of an error that you were asked to test.
- You can use any built-in function if it is not prohibited in the exercise.
- All your functions must have a documentation (___doc___)
- Your code must be at the norm
  - pip install flake8
  - alias norminette=flake8

Chapter III

Exercise 00

	Exercise 00
Exercice 00: Give my BMI	
Turn-in directory : $ex00/$	
Files to turn in : `give_bmi.py`	
Allowed functions : `numpy or any lib of table manipulation`	

Your function, give_bmi, take 2 lists of integers or floats in input and returns a list of BMI values.
Your function, apply_limit, accepts a list of integers or floats and an integer representing a limit as parameters. It returns a list of booleans (True if above the limit).
You have to handle error cases if the lists are not the same size, are not int or float...

The prototype of functions is:

```
def give_bmi(height: list[int | float], weight: list[int | float]) -> list[int | float]:
  #your code here

def apply_limit(bmi: list[int | float], limit: int) -> list[bool]:
  #your code here
```

Your tester.py:

```
from give_bmi import give_bmi, apply_limit

height = [2.71, 1.15]
weight = [165.3, 38.4]

bmi = give_bmi(height, weight)
print(bmi, type(bmi))
print(apply_limit(bmi, 26))
```

Expected output:

```
$> python tester.py
[22.507863455018317, 29.0359168241966] <class 'list'>
[False, True]
$>
```

Chapter IV

Exercise 01

	Exercise 01
Exercice 01: 2D array	
Turn-in directory : $ex01/$	
Files to turn in : `array2D.py`	
Allowed functions : `numpy or any lib of table manipulation`	

Write a function that takes as parameters a 2D array, prints its shape, and returns a truncated version of the array based on the provided start and end arguments.
You must use the slicing method.
You have to handle error cases if the lists are not the same size, are not a list ...

The prototype of function is:

```
def slice_me(family: list, start: int, end: int) -> list:
  #your code here
```

Your tester.py:

```
from array2D import slice_me

family = [[1.80, 78.4],
          [2.15, 102.7],
          [2.10, 98.5],
          [1.88, 75.2]]

print(slice_me(family, 0, 2))
print(slice_me(family, 1, -2))
```

Expected output:

```
$> python test_array2D.py
My shape is : (4, 2)
My new shape is : (2, 2)
[[1.8, 78.4], [2.15, 102.7]]
My shape is : (4, 2)
My new shape is : (1, 2)
[[2.15, 102.7]]
$>
```

Chapter V

Exercise 02

	Exercise 02
Exercice 02: load my image	
Turn-in directory : $ex02/$	
Files to turn in : `load_image.py`	
Allowed functions : `all libs for load images and table manipulation`	

You need to write a function that loads an image, prints its format, and its pixels content in RGB format.
You have to handle, at least, JPG and JPEG format.
You need to handle any error with a clear error message

Here's how it should be prototyped :

```
def ft_load(path: str) -> array: (you can return to the desired format)
  #your code here
```

Your tester.py:

```
from load_image import ft_load

print(ft_load("landscape.jpg"))
```

Expected output:

```
$> python tester.py
The shape of image is: (257, 450, 3)
[[[19 42 83]
  [23 42 84]
  [28 43 84]
    ...
  [ 0  0  0]
  [ 1  1  1]
  [ 1  1  1]]]
$>
```

Chapter VI

Exercise 03

	Exercise 03
Exercice 03: zoom on me	
Turn-in directory : $ex03/$	
Files to turn in : `load_image.py, zoom.py`	
Allowed functions : `all libs for load, manipulate, display image and table manipulation`	

Create a program that should load the image "animal.jpeg", print some information about it and display it after "zooming".

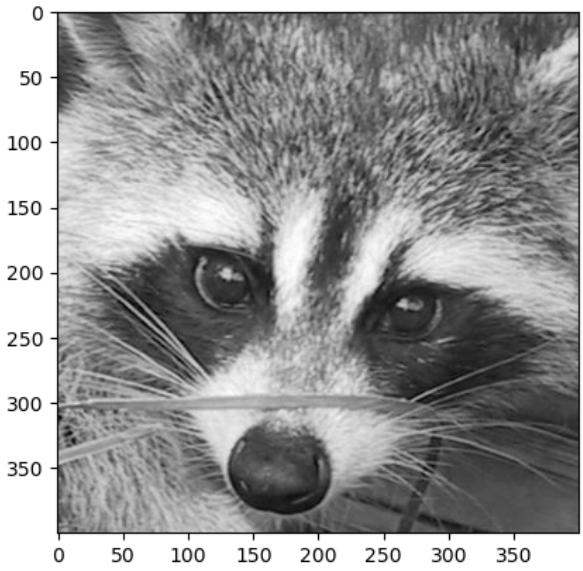
- The size in pixel on both X and Y axis
- The number of channel
- The pixel content of the image.
- Display the scale on the x and y axis on the image

If anything went wrong, the program must not stop abruptly and handle any error with a clear message.

Expected output:

```
$> python zoom.py
The shape of image is: (768, 1024, 3)
[[[120 111 132]
[139 130 151]
[155 146 167]
    ...
[120 156 94]
[119 154 90]
[118 153 89]]]
New shape after slicing: (400, 400, 1) or (400, 400)
[[[167]
  [180]
  [194]
  ...
  [102]
  [104]
  [103]]]
$>
```

Expected output:



Your array after slicing and the zoom area may be different.

Chapter VII

Exercise 04

	Exercise 04
Exercice 04: rotate me	
Turn-in directory : $ex04/$	
Files to turn in : `load_image.py, rotate.py`	
Allowed functions : `all libs for load, manipulate, display image and table manipulation`	

Make a program which must load the image "animal.jpeg", cut a square part from it and transpose it to produce the image below. It should display it, print the new shape and the data of the image after the transpose.

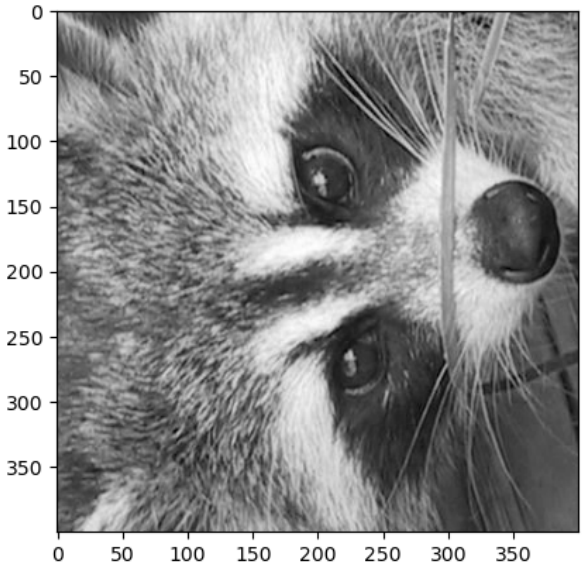
Expected output:

```
$> python rotate.py
The shape of image is: (400, 400, 1) or (400, 400)
[[[167]
  [180]
  [194]
  ...
  [102]
  [104]
  [103]]]
New shape after Transpose: (400, 400)
[[167 180 194 ... 64 50 72]
 ...
 [115 116 119 ... 102 104 103]]
$>
```

Your array after the transpose can be different.
You can look for the transpose method, it could help you ...

`You have to do the transpose yourself, no library is allowed for the transpose`

Expected output:



Chapter VIII

Exercise 05

	Exercise 05
Exercice 05: Pimp my image	
Turn-in directory : $ex05/$	
Files to turn in : `load_image.py, pimp_image.py`	
Allowed functions : `all libs for load, manipulate, display image and table manipulation`	

You need to develop 5 functions capable of applying a variety of color filters to images, while keeping the image shape the same.

Here's how they should be prototyped :

```
def ft_invert(array) -> array:
  #your code here

def ft_red(array) -> array:
  #your code here

def ft_green(array) -> array:
  #your code here

def ft_blue(array) -> array:
  #your code here

def ft_grey(array) -> array:
  #your code here
```

You have some restriction operators for each function: (you can only use those given, you don't have to use them all)

- `invert`: =, +, -, *
- `red`: =, *
- `green`: =, -
- `blue`: =
- `grey`: =, /

Your tester.py:

```
from load_image import ft_load
from pimp_image import ft_invert
  ...

array = ft_load("landscape.jpg")

ft_invert(array)
ft_red(array)
ft_green(array)
ft_blue(array)
ft_grey(array)

print(ft_invert.__doc__)
```

Expected output: (docstrings can be different)

```
$> python tester.py
The shape of image is: (257, 450, 3)
[[[19 42 83]
  [23 42 84]
  [28 43 84]
    ...
  [ 0  0  0]
  [ 1  1  1]
  [ 1  1  1]]]
   ...
Inverts the color of the image received.
$>
```

Expected output: (you must display the images transformed)

Figure VIII.1: Original

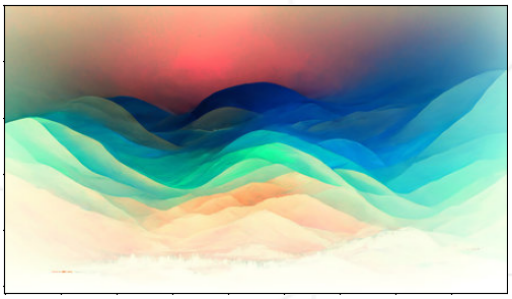


Figure VIII.2: Invert

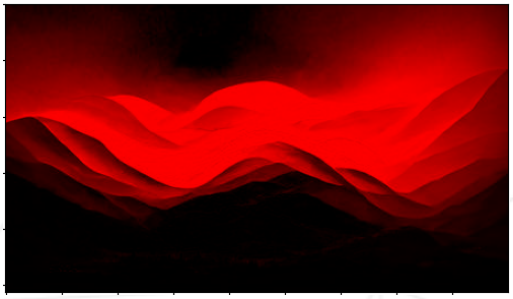


Figure VIII.3: Red

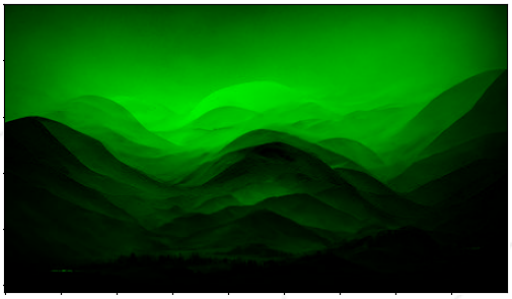


Figure VIII.4: Green

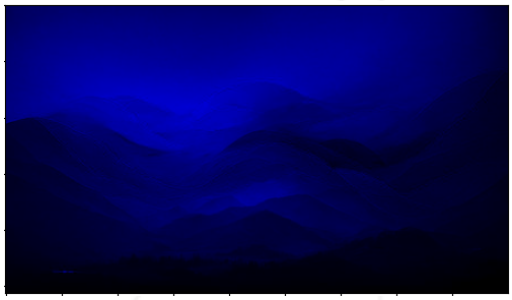


Figure VIII.5: Blue

Figure VIII.6: Grey

Chapter IX

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

`The evaluation process will happen on the computer of the evaluated group.`