Exercise Set 4
More Python

University of Oslo - INF3331/INF43331

Fall 2018

Note: These exercises are not mandatory. You don't have to put your work into your Github repository.

Exercise 1: Making images

Exercises in Python can be represented as 3D-arrays with shape (H, W, C) where H is the number of rows, W the number of columns, and C the number of channels (3 in the case of RGB images).

Create a grayscale image where every pixel value is generated randomly (either floats between 0 or 1, or integers between 0 and 255). Look at your image (for instance using `matplotlib.pyplot.imshow()`)

Take your image, and split the values into three groups, e.g. with values in (0, 0.33), [0.33, 0.66) and [0.66, 1]. Color each of these different colors by. You should be able to do this purely in Numpy. Loops should be avoided, as they are slower.

Exercise 2: Decorators

Assume that `f` is a Python functions. Because functions are objects in Python, we can pass functions to other functions. It turns out that writing

```
f = some_func(f)
```

Is quite common. There is a shorthand for this. When writing the definition of `f`, we can say

```
@some_func
def f():
    # Whatever f does
```

We have applied a decorator to `f`. An example of a decorator is one that prints the output of f before it is returned. It can be implemented in the following way:

```
def print_output(f):
    def wrapper(x):
        output = f(x)
        print(output)
        return output
    return wrapper
```

Using this decorator does not change the behaviour of `f`, except that the output is always printed when f is called.

Exercise: Make a decorator that tracks the time a function uses to execute

Decorators can be (and usually are) implemented as classes, where the class' `__call__` method is used as the wrapper. This can be useful if we want to keep some information between calls. We are going to try this next.

Exercise: Create a decorator that caches output from f. i.e., before calling `f`, check if we already know what the return value is, and return this if this is the case. This is useful if `f` takes a long time to execute.