Mandatory Assignment 4

Basic Python programming (30/40 + 5 bonus points points)

University of Oslo - INF3331/INF43331

Fall 2018

Your solutions to this mandatory assignment should be placed in the directoryassignment4in your Github repository.Your delivery should contain a Readme file containing information on howto run you scripts. In particular, it is important that you document how torun your tests.In addition, your code should be well commented and documented. Allfunctions should have docstrings explaining what the function does, how, andan explanation of the parameters and return value (includeing types). Yourcode should also be well formatted and readable.

4.0 Mathematical background (0 points)

In this assignment, you will make a program which plots a fractal called the Mandelbrot set. For an explanation of how to draw it, check out out `https://plus.maths.org/content/unveiling-mandelbrot-set` We will use $\mathbb{C}$ to refer to the complex numbers. If you are unfamiliar with complex numbers, think of them as the normal plane of all points (a, b) with an operation called complex multiplication $(a, b) \cdot_{\mathbb{C}} (c, d) = (ac - bd, ad + bc)$.

For each complex number c, define a function[1] $f_c(z) = z^2 + c$. For some numbers c, the sequence $0, f_c(0), f_c(f_c(0)), f_c(f_c(f_c(0))), \ldots$ will grow towards infinity. For others, it will not. The *Mandelbrot set* is the set of all c for which this sequence does `not` tend towards infinity. It is known (and you may assume) that if at any point the sequence is above 2, it tends towards infinity (and if not, it does not). So for example, you can check that (1, 0) is not in the Mandelbrot set, but (-1, 0) is.

To draw a picture of the Mandelbrot set, then, you can color each complex number c as black if it is in the Mandelbrot set, and otherwise color it according to how many times you need to apply f_c to 0 to get above 2. In practice, the nicest way to do this is to choose some fine, regularly spaced grid of points in the complex plane, each corresponding to a pixel in the image you are drawing, and

[1] In coordinates, if $z = (x, y)$ and $c = (a, b)$, $f_c(z) = (x^2 - y^2 + a, 2xy + b)$

for each point compute elements of the sequence $0, f_c(0), f_c(f_c(0)), \ldots$ until you either go above 2, in which case you note how many steps it took, or until you reach some preset threshold[2] (say 1000) steps, in which case you conclude it's probably in the Mandelbrot set. Then, use some color scale mapping numbers between 0 and 1000 to colors and color the pixels accordingly.

Note: You should *not* use your Complex class from assignment 3. Instead, use Python's built-in class.

4.1 Python implementation (6 points)

First, create a Python script which chooses some rectangle in the complex plane, colors that rectangle according to the method above, and saves the coloring as an image. You do not have to make a user interface for your script yet, but it should be easy to change the area drawn by modifying parameters inside the script.

You *are* allowed to use numpy for this part of the problem, but only for storing numbers. All computations should be done with pure python

Name of file: `mandelbrot_1.py`

4.2 Numpy implementation (6 points)

Make a similar script to 4.1 so that all computationally heavy bits use numpy arrays. Compare the runtime on some input of your choosing. How much is gained by switching to numpy? Your report should contain the parameters used to generate the images, along with the runtime for each script.

Name of file: `mandelbrot_2.py`, `report2.txt`

4.3 Numba implementation (6 points)

Redo 4.1, but use Numba to speed it up. Compare the runtime as before. Can you think of any advantages/disadvantages to using Numba instead of Numpy?

Name of file: `mandelbrot_3.py`, `report3.txt`

4.4 Cython implementation (INF4331 only: 10 points)

Redo your impliementation in Cython, and create a report as before
Name of files: `mandelbrot_4.py`, `report4.txt`

[2] We would like to check an infinite number of iterations, but unfortunately we don't have time for that.

4.5 4.5: User interface (6 points)

Provide a command line user interface for your script. The design of this is up to you, but it should provide instructions for its by calling it with a `--help` flag, and it should be possible to specify which region of the plane to draw, the resolution to draw in and output image filename. It should also be possible to switch between your 3 implementations with a command line argument.

Name of file: `mandelbrot.py`

4.6 4.6: Packaging and unit tests (6 points)

Make your implementation into a Python module or package and make a setup script. Your package should include a method `compute_mandelbrot(xmin, xmax, ymin, ymax, Nx, Ny, max_escape_time=1000, plot_filename=None)` which returns an $Nx \times Ny$ array of the escape times of evenly sampled points in the rectangle [xmin, xmax] $\times$ [ymin, ymax]. If `plot_filename` is supplied, an image should be rendered and saved to the specified location. You are free to make other methods which you think add useful functionality.

Further, make two unit tests. One should test that if you choose a region of the plane which is entirely outside the Mandelbrot set after 0 iterations (like the rectangle [3, 4] $\times$ [3, 4]), your script notices. Another should check that if you take a region which is entirely inside the Mandelbrot set, your script should not say they are outside. You are free to choose which unit testing framework to use for writing your tests, provided it is reasonably standard. py.test is a good option.

Name of files: `test_mandelbrot.py`, `setup.py`

4.7 4.7: More color scales + art contest (2 bonus points)

Use your script to make pretty pictures. Add support for additional color scales to your script, giving the user at least 3 visually distinct options, and experiment with different color scales and locations to render, and make up a picture you think look cool. The picture deemed nicest will win a small surprise. (The contest is in addition to the bonus points. Simply submitting will give you the points)

Name of your contest submission: `contest_image.jpg/png/whatever`

4.8 4.8: Good and bad Python code (3 bonus points)

Create two functions (`repeat_bad` and `repeat_good`). They should both take in a string, and return a string. What they should do with the string is best demonstrated by examples:

- `abcd => A-Bb-Ccc-Dddd`
- `abcBd => A-Bb-Ccc-Bbbb-Ddddd`

Every character is repeated according to it's position in the original string. The first occurence is always capitalized, and the other ones are always lower case. The groups of characters are joined by hyphens.

The two functions should behave exactly the same, but one should be written in the most ugly way you can come up with. i.e. break both Python programming conventions, and general conventions. Also try to solve the problem in a strange way.

In the other functions, you should try to write good Python code. Try to use the tools Python gives you.

Name of file: `replicator.py`