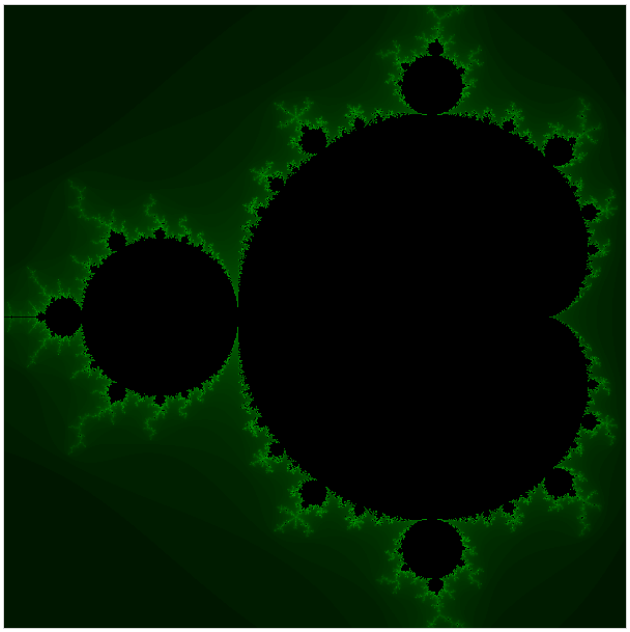
2IS50: Homework Assignment 1 – Mandelbrot

13 April 2021

The link to this assignment on GitHub Classroom will be made available separately. First, make sure that your project is set-up correctly, like Homework Assignment 0. See the file `README.md` of this assignment in your own repository.

1 Mandelbrot Set

In mathematics, the Mandelbrot set is a fractal (see figure below). This means you can zoom in indefinitely and the same patterns will occur. In this assignment, you will be generating images of the Mandelbrot set yourself.



1.1 Mathematics

Before looking at the Mandelbrot set, we will have a look at the *Mandelbrot number*, or *mandel-number* for short, of a point (x, y) in the xy-plane. Define the *mandel sequence* (u_n, v_n) for point (x, y) by

$$(u_0, v_0) = (0, 0) \tag{1}$$
$$(u_{n+1}, v_{n+1}) = (x + u_n^2 - v_n^2, y + 2u_n v_n) \tag{2}$$

By definition (the reason does not matter), the point (x, y) has mandel-number n, when n is the smallest value for which $u_n^2 + v_n^2 > 4$. We say that the Mandel sequence *diverges* at that value of n.

For example, $(1, 0)$ has mandel-number 3, since it produces the sequence

$$\begin{array}{cccc} (u_0, v_0) & (u_1, v_1) & (u_2, v_2) & (u_3, v_3) & \dots \\ = (0, 0), & = (1, 0), & = (2, 0), & = (5, 0), & \dots \end{array} \tag{3}$$

and $5^2 + 0^2 > 4$.

This sequence can go on indefinitely without diverging, that is, without ever encountering $u_n^2 + v_n^2 > 4$. Thus, we want to bound it when we compute it. A reasonable bound is that we calculate the first 100 elements of the sequence, and just assume it has an infinite Mandel number if it didn't diverge yet.

The Mandelbrot set is defined as all those points (x, y) with an infinite Mandel number. For a colourful picture of the Mandelbrot set, you can let the colour depend on the Mandel number. Moreover, it is interesting to zoom in on specific areas of the Mandelbrot set. That is what we will pursue in this assignment.

1.2 Initial repository

We've provided three source files in the `src/` directory. In the file `mandel.py` you should define the functions, which we already named. The file `gui.py` provides a basic Graphical User Interface (GUI), such that we get results on the screen. It is made using PyQt5. Finally, the `main.py` file calls the functions defined in the other files and has the main loop that keeps the interface running.

Note: We provided the type hints for all the given functions. If you create any additional functions, they should have proper type hints as well.

1.3 Pair programming

IMPORTANT: This assignment should be done with pair programming, where the roles of driver and navigator are regularly switched. This NEEDS to be reflected in the history of the repository. Both authors need to have created/closed issues, and both need to have committed in the various stages of this assignment. Thus, commits should be interleaved between the different authors. Failing to do so, WILL cost you points.

1.4 Grading

In total, you can get more than 100 points (65 for basics, 70 for extra), but the maximum points you can get is 100. Thus, you can do extra exercises to compensate for small mistakes, but it is not required to do all the extras. Although, you cannot compensate for exercises related to software engineering. (exercise a, b, and d). Not doing at least exercise a and some useful commit before the first deadline costs you 5 points.

The grade is determined by

$$Grade = \frac{\min(Basic + Extra, 80) + Mandatory - Penalties}{10} \tag{4}$$

Basic = Exercise c + e + f + g
Mandatory = Exercise a + b + d
Extra = Exercise i + j + k + l + m + n + o + p
Penalties = **if** missed deadline **then** 5 **else** 0 + **if** not (pair programmed) **then** 5 **else** 0

2 Basic Exercises

Do the following exercises to get a passing grade. To get a higher grade, you should do some extra exercises.

Note: As a first milestone, finish the following basic exercises *in the first week*.

Exercise a (5 points)
In your repository, create:

- One issue about the basic exercises, list everything that needs to be done in a list.
- One issue about the extra exercises, list everything that could be done.

Assign each issues to a different person (although you are both responsible). Keep the issues up to date, with the progress you make.

Exercise b (5 points)
Generate the documentation and copy `docs/build/` to `docs/html/` and add it to the repository. Ensure that in the end all functions have complete documentation (a general description, description of their parameters and type information) and you regenerate the documentation, and copy it again to `docs/html` and commit it.

Exercise c (10 points)
Define the function called `mandel_seq`, which returns the mandel sequence, up to a certain number (n) or when it diverges. Also define function `mandel_number` which returns the mandel-number.

Exercise d (10 points)
In the next two exercises, you have to define the functions `convert_pixel` and `color_mandel`. Before implementing them, add a correct `docstring` to both explaining what the function and parameters do, and add two `doctest` examples to both.

Exercise e (10 points)
Before we can visualize the Mandelbrot set, we have to convert pixel coordinates to actual coordinates in the xy-plane. Assume that the x range is $[-1.5, 0.5]$, the y range is $[-1, 1]$ and that the image has size 600×600. Define the function `convert_pixel`.

Exercise f (15 points)
We now want to put the Mandelbrot image on the actual screen. In the file `gui.py` we already provided some helper functions that do the visual work for you. The given code puts some squares on the screen. You should define function `color_mandel`, which should be called in the function `make_image` (from the GUI class).

Note: You want to give each pixel a colour. Give pixels that diverge immediately and ones that do not diverge after n iterations both the colour black. The numbers in-between can be given a colour by scaling with the Mandel number. A linear scaling would seem logical, but actually, our eyes work differently. Therefore, a scaling based on the square root (or logarithm) of the Mandel number works better.[1]

Exercise g (10 points)
Add four `pytest` functions in the `tests/test_cases.py` file. They should test the functionality of the above functions, that has not been tested yet, and which is sensible to test.

[1] The same is true for sound, that's why we measure sound in decibels, which is a log scale.

3 Extra Exercises

To get a higher grade, do some additional exercises below. Any added functions or parameters to (existing) functions should be documented. Keep the generated documentation up to date (in `docs/html`).

For a tutorial on using PyQt5, you can consult: *PyQt5 Tutorial*.

The official documentation of PyQt5 is available here: `https://www.riverbankcomputing.com/static/Docs/PyQt5/index.html`. The Qt5 documentation is better & more complete, (PyQt5 is just a binding to Qt5), which you should also consult: `https://doc.qt.io/qt-5/index.html`

Exercise h (5 points)
Use radio buttons to allow the user to choose different colour schemes for the Mandelbrot set.

Exercise i (5 points)
Add a field, which allows the user to provide a new centre point for the Mandelbrot set.

Exercise j (10 points)
Add a field, which allows the user to provide a new range to show for the Mandelbrot set. The range for the x and y coordinates so far is fixed at $[-1.5, 0.5]$ for x and $[-1, 1]$ for y (in `convert_pixel`). Allow this to be $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$ for any values.

Exercise k (10 points)
Since the Mandelbrot set is a fractal, it can be zoomed in indefinitely, and still, have a structure. Make a zoom function, which responds to mouse clicks. The left mouse click should zoom in, the right mouse click should zoom out, and the position of the mouse should be the new centre of the image. Also, make a reset button.

Note: To get the information about mouse clicks, most graphic user interfaces work with events, the same is true for the PyQt5 library. We've already set up the `mousePressEvent` method for you which receives an QMouseEvent object, read about the `QMouseEvent`[2] which contains all the information you need.
Important: Zooming in should show more detail, not simply scaling up pixels.

Exercise l (10 points)
Add an extra button that allows you to save the Mandelbrot image. Users should be able to specify the name of the saved file.

Note: Somewhere we are already using the QPixmap class in the code. Look in the documentation of the QPixmap class of the PyQt5 library[3] for functionality for saving images. To get the QPixmap value of a QLabel, you can use the `pixmap()` method.

Exercise m (10 points)
A very simple filter to alter photos is the blur filter. It takes the average value of the neighboring pixels of an image, to calculate the new value. E.g., a horizontal blur filter would look like:

```
imageblur[x, y] = (image[x-1][y] + image[x][y] + image[x+1][y]) / 3
```

Implement a blur filter that works on the Mandelbrot image. It should be invoked by an extra button that is added to the user interface.

[2] `https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtgui/qmouseevent.html` and `https://doc.qt.io/qt-5/qmouseevent.html`

[3] `https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtgui/qpixmap.html` and `https://doc.qt.io/qt-5/qpixmap.html`

Exercise n (10 points)
You may have noticed that the creation of the Mandelbrot image takes quite some time. Part of this is because Python is an interpreted language: it doesn't compile the code beforehand. Have a look at *Numba*[4], a just-in-time compiler for Python, and see if you can make the calculations faster.
Note: This can be very subtle to get right, please ask for help if strange errors occur.

Exercise o (10 points)
Make a Newton fractal[5] for the complex function $p(z) = z^3 - 1$. The user should be able to switch between the Mandelbrot fractal and the Newton fractal via the GUI.

[4] `http://numba.pydata.org/`
[5] See wikipedia: `https://en.wikipedia.org/wiki/Newton_fractal`