PHYS 210, Assignment 11

Create a new directory somewhere in your home directory with the name `yourusername_assignment_11` to store the files you will create for this assignment. To hand the assignment in, copy the directory with your results to `/home2/phys210/yourusername/`. Make sure it's there and has the right permissions (read and execute for everyone, write for you).

1 Dictionaries

Write all the functions below in a script dictionaries.py. None of your functions should generate an error if called with an empty dictionary.

1. Write a function `concatenate(a,b,c)` which returns the concatenated dictionaries `a,b,c`.
2. Write a function `exists(d,s)` which checks if key `s` already exists in dictionary `d`.

2 Broadcasting

Which lines of the following lines are valid code? Which are not? How would you fix them? Put your answer in a filed called `broadcasting.txt`.

```
a = np.arange(5)
b = np.random.random(4)
c = np.arange(20).reshape((4,5))

d = a + b
e = a + c
f = b * c
g = c.T * b
```

How would the results change if `a`, `b`, and `c` where matrices instead of `numpy` arrays?

3 Mandelbrot III

We now have all the tools to produce pictures similar to figure 1 on assignment 6.

1. Create an array of complex numbers x in the range $-2.5 < \Re(x) < 1$ and $-1.5 < \Im(x) < 1.5$, using for example `numpy.meshgrid`. To start with, use a coarse resolution like 100 on the long side.
2. Modify your function from assignment 7 to return the number of iterations needed for z_n to diverge. The numbers of iterations can be used to colour the numbers not in the set.

3. For each complex number of the grid created in the first part of this exercise, compute the numbers of iterations it takes for the Mandelbrot series to diverge. Use a low cut-off for the maximum number of iterations for now, e.g., 100. Plot the resulting array using `matplotlib.pyplot.imshow`. Save the file as `mandelbrot_lowres.pdf`

4. To create pretty high-resolution images in a reasonable time, the code has to be optimized a bit. You have already seen that replacing `for` loops by array operations can speed up the code quite dramatically. Write a function `mandelbrot_set` that takes an array of complex numbers c and the maximum number of iterations and returns the number of iterations necessary for z_n to diverge.

5. Your code should now be fast enough to produce nice pictures reasonably fast. Make a plot of the full Mandelbrot set and save it as `mandelbrot_highres_1.pdf`. Make another plot of a part of the set of your choosing (see figure 1 for an example) by changing the range of complex numbers in the first part of this exercise. Save it as `mandelbrot_highres_2.pdf`.

Here are some hints if you want to further play around with this:

- To show the colour gradients better, especially for high numbers of iterations, it can be helpful to scale the results by taking the logarithm or square root.
- The colour scale can be specified by the `cmap` keyword. I used `magma_r` for the example figure.
- To create smooth colour gradients it is necessary to interpolate the output of your function. Take a look at `https://en.wikipedia.org/wiki/Mandelbrot_set#Continuous_.28smooth.29_coloring` to learn how to do this.

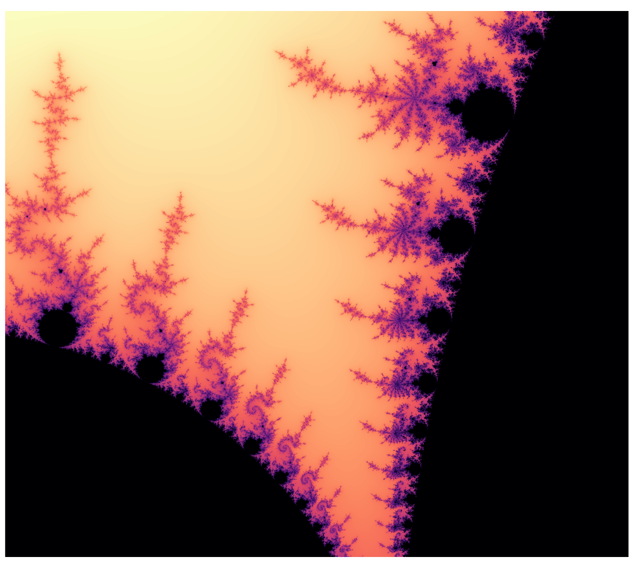

Figure 1: The Mandelbrot set around $c = -1.26 + 0.06\imath$.