# Homework assignment week 2

# GitHub Instructions

1. Create a new repository where all team members are collaborators. The name of the repository should be ds5\_assignment\_ group#, where # is your groupnumber. You will use this repository for the other group homework assignments as well.
2. Add Roos, Babette and Jelle to the repository as collaborator as well (username: reherrings, bamlips, jellebosschaart)
3. Create one .ipynb (or .py) file that contains the answers to all the exercises below! However, you can call other files in your main file.
4. Make sure that you push and commit all your changes to the file before the deadline. If changes are made after the deadline, it will not be taken into account for the bonus point

# Exercise 1: Structuring a given code

You have been given a code (see below) that reads a CSV file containing student records, performs various calculations and generates a report after some data manipulation. However, the code is complex, difficult to understand and missing documentation.

For this exercise, perform the following tasks:

1. Work in Github from the start
2. Break down the code into smaller functions
3. Add descriptive docstrings to each function and type hints for your functions

Note, even though you do not have the student records available, you should be able to break a working code up and add descriptive docstrings.

Code:

file\_path = input("Enter the path to the CSV file: ")

records = []

with open(file\_path, 'r') as file:

    csv\_reader = csv.DictReader(file)

    for row in csv\_reader:

        records.append(row)

total = sum(float(record['Grade']) for record in records)

average = total / len(records)

print(f"Average Grade: {average}")

print("--------------------")

filtered\_records = [record for record in records if float(record['Grade']) >= 80.0]

print("Student Report")

print("--------------")

for record in filtered\_records:

    print(f"Name: {record['Name']}")

    print(f"Grade: {record['Grade']}")

    print("--------------------")

# Exercise 2: Mandelbrot visualisation

The Mandelbrot set is a set of complex numbers defined by the sequence

with complex number . The number is part of the Mandelbrot set when does not diverge to infinity. In python, is often used. This Mandelbrot set is a fractal and can be visualized as in figure 1. We will be generating such an image. The black area of the image correspond to the number for which did not diverge to infinity.

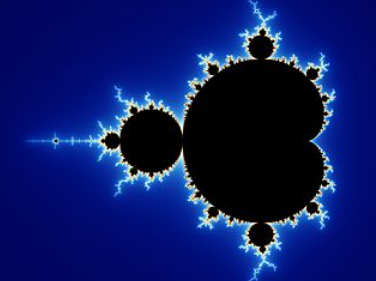


Figure 1: Visualisation of the Mandelbrot set

Normally, we would see if the sequence would diverge for , but we do not have enough time to check it for infinite numbers. For this reason, we assume that the sequence diverges to infinity if and we call this the *diverging index*. If we cannot find a diverging index less than 100, then we assume that the sequence diverges to infinity and the diverging index will be defined as 0.

Your assignment is to draw a 200x200 image of the Mandelbrot set. The x range is [-1.5, 0.5] and the y range is [-1, 1]. Make a function draw\_mandel(width) where width is the width and height of the image (so it is always a square!). Use the diverging index as a colour representation of the pixel color.

To start, divide this exercises in pieces. Make a list of functions (not a Python list) that needs to be made. Clearly define what the purpose of a function is and what its inputs and outputs are. If your tasks are clear, you can start coding. Communicating and collaboration for this assignment might be harder than the programming itself.

Things to take into account:

1. Work in Github from the start (mandatory)
2. Create docstrings and type hints for your functions (mandatory)
3. A numpy array can often be used as a representation of an image. With matplotlib, you can visualise that image. Let someone do some research on working with images in Python.
4. Note that the assignment is to draw a 200x200 image, but your function should be generic for any squared image. Test your image also for 300x300, 500x500 and 800x800.

In the end, you should have a draw\_mandel(200) function call in your code.

# Exercise 3: Google PageRank algorithm and the worldwide web

## Introduction PageRank

PageRank algorithm is the algorithm that Google uses/used to rank webpages for their search engine. The algorithm is named after the Google founder Larry Page. This algorithm models the odds that someone lands on a webpage by randomly clicking hyperlinks in webpages. We can approximate the PageRank by simulating that process.

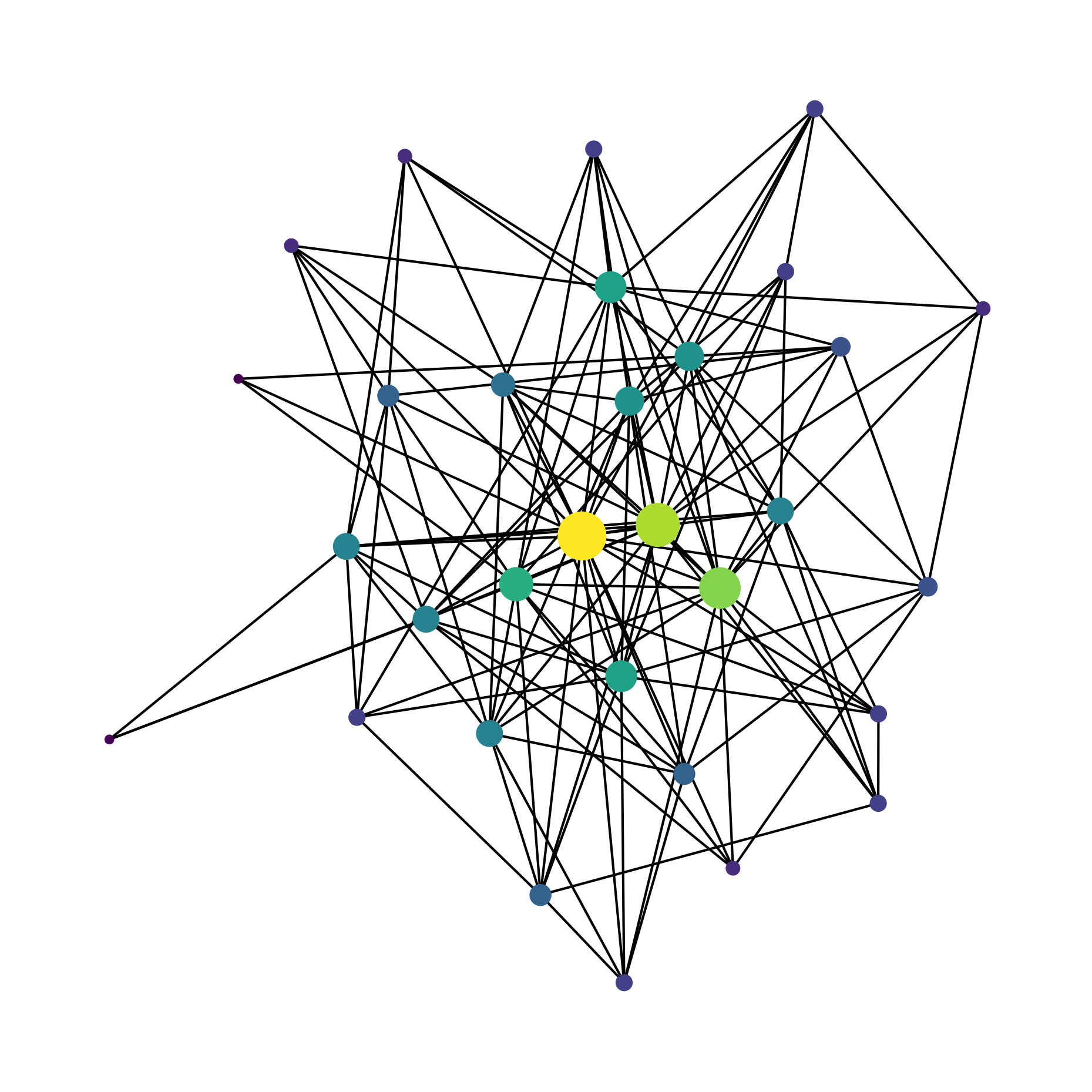
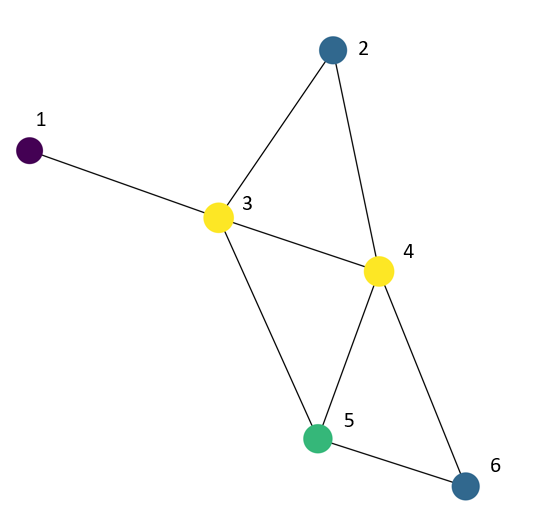


Figure 2: Example of a generated network

Let’s look at a small example:



You are surfing on the web randomly and are currently on page 3 (node 3 in above graph). With probability , we go to a randomly chosen page that is linked to page 3. This means that we randomly chose page 1, 2, 4 or 5, as those are all connected to page 3 (see the graph again). All linked pages (1, 2, 4 and 5) have equal probability to be chosen.

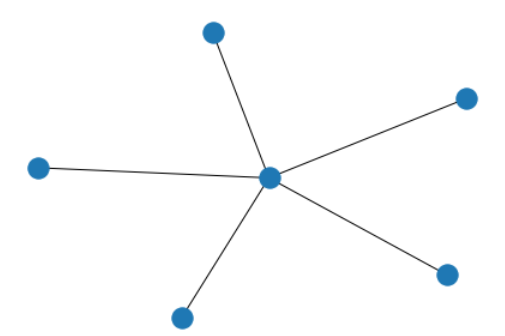
With probability we go to any random page that is not connected to the page we are currently visiting. In above example, where we are visiting page 3, this means that there is probability that we surf to page 6 (by typing in the URL of page 6 in the address bar).

For each page, we count the number of times we visited that page. In total, we visit number of pages. The estimated PageRank of every page is the number of visits to that page divided by .

## Introduction Random Networks

In order to calculate the PageRank, we first need to have a graph that represents the world wide web. To work with graphs, we use the package networkx. The network we are going to make is based on the Barabasi-Albert algorithm.

We start with a star network that we can generate with networkx.star\_graph(n0). This makes a graph with n0 nodes around 1 ‘hub’ (a center nodes that is connected to all the other nodes, see below image where we used n0 = 5). We often see this behaviour on the world wide web, where multiple websites link to 1 website (for example Wikipedia or YouTube).



Next, someone creates a new webpage and links to already existing pages of the star network with probability

Here, is the probability that page links to page . Furthermore is the number of links page already has (this is often called the *degree* of ). This means that, the more links a webpage has, the greater the probability that a new page will link to that page.

In total we want webpages where every webpage links to *different* pages. We start with the star graph from the example with .

*Note: pages cannot link to themselves.*

## Exercise 3.1

Model a Barabasi-Albert network as described in the introduction. Calculate the PageRank on this network. Make a notebook where you visualise your generated network and the distribution of the PageRank ‘probabilities’.

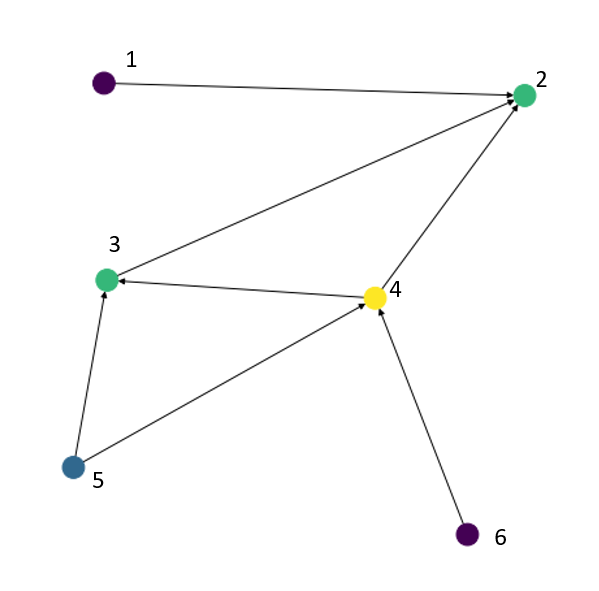
Things to think about:

1. Work in Github from the start (mandatory)
2. Create docstrings and type hints for your functions (mandatory)
3. Generate your network with networkx (Python package), but you are not allowed to use the graph generators (mandatory). First, make a small network to learn networkx and then try to implement the algorithms. A good exercise would be to make the star graph manually and visualise it.
4. We gave requirements for , and , but your code should work for general and .

## Exercise 3.2

In exercise 3.1, we modelled an undirected network. This means that if node is linked to node , then node is automatically also linked to node . However, this is not so realistic: the probability that a new webpage links to Wikipedia is big, but the probability that Wikipedia links to that new webpage is very small. We have to adjust our model so that it becomes a directed model.

A directed graph looks like this:



Now for example, as you can see from the way the arrow is positioned, page 3 links to page 2, but page 2 does not link to page 3.

Let’s look at a more realistic example. We are going to look at a real network of Wikipedia, namely the network of squirrel articles. Why you might ask? The answer is contained in the next image.



Figure 3: Why not?

In ‘squirrel\_edges.csv’, we see the connections between all the nodes. In the left column, we can observe the source article, while in the right column, we can see the target article being referenced.

Make a directed graph of this network (you are allowed to use nx.DiGraph() to create a directed graph). Furthermore for the PageRank algorithm, we are no longer able to visit arbitrary pages linked to our current page. Instead, we are restricted to accessing only the pages that are directly linked. Modify your code such that it works for directed and undirected graphs.

In the same notebook as exercise 3.1, calculate the PageRank of this network and visualise the distribution of the PageRank ‘probabilities’. To calculate the PageRank of a network, you can have a look at the pagerank\_numpy function (<https://networkx.org/documentation/networkx-1.8/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank_numpy.html>) .