In this homework, you will visualize the network of a population of wild badgers from the Weber et al. paper. Feel free to use components of Lab 2 and collaborate with others, as long as these things are cited in your code.

1 Read the Badger Matrix

The file `BadgerMatrix.txt` contains a table that represents, for every pair of badgers, the number of seconds that the badgers were in contact (we'll call it *contact time*) throughout one year. The first few rows and columns of this file look like the following:

```
Badger 008p 009p 010p 010y 011y 012b ...
008p 0 0 0 0 0 354 ...
009p 0 0 8 0 102688 0 ...
010p 0 8 0 0 3 0 ...
010y 0 0 0 0 0 0 ...
011y 0 102688 3 0 0 766 ...
012b 354 0 0 0 766 0 ...
...
```

Write a function to parse this file and return the adjacency matrix and a list of nodes:

`read_matrix()`: *Read* ***`BadgerMatrix.txt`*** *and return the adjacency matrix and node list*
Inputs: None (or filename if you want to generalize the function).
Returns: A list of strings representing nodes (e.g., `['008p','009p','010p',...]`) and a list of lists representing the adjacency matrix.

Your matrix can be a binary matrix (only 1's and 0's) or a matrix that contains the contact time for all badger pairs (contact time is used in the optional challenge).

1.1 Check if the Adjacency Matrix is Symmetric

The badger network represents an undirected graph, meaning that the adjacency matrix should be symmetric. A matrix is *symmetric* if `A[i][j]=A[j][i]` for all pairs of indicies `i` and `j` in the matrix. Write a function that checks if an adjacency matrix is symmetric:

`check_symmetric()`: *Check if an adjacency matrix is symmetric.*
Inputs: Adjacency matrix `A`.
Returns: `True` if `A` is symmetric; `False` otherwise.

Confirm that the matrix is symmetric in your code (print a `yes` or `no` statement depending on the output of `check_symmetric()`).

1.2 Create a List of Edges

Write a function that creates a list of edges from the adjacency matrix and the node list:

`mat_to_edgelist()`: *Convert an adjacency matrix to an edge list.*
Inputs: Adjacency matrix `A` and a node list `L`
Returns: List of 2-element lists describing edges, e.g.,
`[['008p','012b'], ['008p','016p'], ['008p','018p'], ...]`

Hint: *You want each edge to be listed* ***exactly once*** *in the returned edge list.* That is, you don't want to have `['008p', '012b']` and `['012b', '008p']` be different elements in your list.
Hint: *If you want to use sets instead of lists, consider storing 2-element tuples.* Lists are mutable, meaning that you can change elements within them. Elements of sets, however, must be immutable (ints, strings, floats, or tuples).

1.3 Post the Graph to GraphSpace

You are now ready to post your graph to GraphSpace. Follow the instructions from Lab 2, using the list of nodes from the `read_matrix()` function and the list of edges from the `mat_to_edgelist()` function.

1.4 Annotate the Graph

The file `BadgerInfo.txt` contains three pieces of information for every badger: (1) badger sex (`Male` or `Female`), (2) infection status (`P` for positive and `N` for negative); and (3) social group (an integer from 1 to 8). Write a function that reads in `BadgerInfo.txt` and returns all three pieces of information for every badger. The type of variable you return is up to you.

`parse_info()`: *Parse `BadgerInfo.txt`.*
Inputs: None (or filename if you want to generalize the function).
Returns: Information about sex, TB status, and social group in a form that you choose.

Once you have this information, add it to the graph. Specifically,

1. Visually denote the **TB status**, **sex**, and **social group** of the badgers.
2. Make the size of each node proportional to the **degree** (the number of neighbors).
3. Add a **popup to the nodes** that lists the sex, TB status, and social group of the badger. The `popup` argument is included when adding nodes and edges, and it takes an HTML string as a value. To format text in HTML, here are some common tags:

   ```
   Regular text is fine. <i>Make this text italic.</i> <b>Make this text bold</b>
   Add a line break <br> between the words ``break'' and ``between''
   ```

   To Get Fancy: `http://www.simplehtmlguide.com/cheatsheet.php`
4. Save a layout that corresponds to the **social group spatial configuration** that is similar to Figure 1 from the paper.
   Note: *The social groups are numbered differently, and the data is slightly different than what's reported in the paper.*

2 Challenge: Incorporate Contact Time (Optional)

You can adjust the edge style to visually denote the **contact time** of the badgers. Here, you will add edge annotations to your existing graph.

1. If you have not already done so, read the file (Section 1) and keep track of contact time for each pair of badgers.
2. Adjust the line width/thickness according to contact time.
   Hint: *You may want to consider scaling the contact time, or converting the contact time to a log scale.*
3. Add a **popup to the edges** that lists the number of seconds that the badgers interacted.
4. Use the Filtering mechanism to **show only the top** t **edges** according to contact time.
   (a) First, look at an example graph that uses filtering (also available under Public Graphs):

   `http://graphspace.org/graphs/23156?user_layout=1896`

   When you load this graph, there is a new side panel called "Filter Nodes and Edges". Observe what happens when you change the sliding bar. The left and right arrows can be used too.

   (b) Each node and edge has an attribute `k`, which is specific to GraphSpace. When adding nodes, include the argument `k=1` for all nodes. When adding edges, include the argument `k=2` for all edges. Post the graph and observe what happens.

   (c) Now, change the value of `k` on the edges so they correspond in some way to the number of seconds. The **lowest** value of `k` should correspond to the badger pair with the **largest** contact time.
   Warning: *Set the maximum value to no more than 500.* You may have to "bin" the edge contact time (adjust how you handle #2) or only consider the *rank* (index) of the edge.

3 Handin Instructions

1. **Add a `HW1` tag & a graph description, and share your final network** with the `BIOL331F19` GraphSpace group.
2. **Document your code.** Add comments, rearrange functions, etc., to make it readable. All code except import statements, global variables, and a `main()` call at the bottom of the file should be contained within functions. You will be graded on the graph and the code.
3. **Submit your code through Moodle** by 1pm on Wednesday.