Lab 5: Object-oriented programming

The labs are getting gradually more difficult, so this fully relies on your capacities as a group, not as an individual! Work together.

Some groups have good programmers in them. Make sure that they do not simply implement everything and others in the group are just watching. Tricks to avoid this (use all!): 1) when working together, have the students who are less comfortable with the programming do the typing; 2) divide tasks and then return to the group to discuss each other's progress; 3) do not quickly implement everything before you meet as a group, but work together.

Refactoring is the technical term of rewriting code that already works, to improve readability and maintainability of the code. For example, sometimes we might split a function into multiple, easier to read functions, that have very clear objectives. So some of the following is refactoring functions we write in earlier labs, by implementing them in a more object-oriented way.

By the end of the lab, this code should work and the values of opinion x for agents should change with each run:

```
from simulation import Simulation

sim = Simulation(agents=10)

sim.print()
sim.run(10)
sim.print()
sim.run(10)
sim.print()
```

The above should be the contents of a file *main.py*.

1. Create an Agent class. Each agent should at least have instance variables for x, ε and τ (see Lab 3). Write the constructor such that the random starting

values for x and ε are also generated.

2. Add a *print()* method to the Agent class that prints basic information about the agent in one line.
3. Create a Simulation class which includes an instance variable that is a list of agents. The constructor should have an input parameter *agents*, which captures the number of agents in the system, and should populate the list of agents accordingly.
4. Add a *print()* method to the Simulation class that prints a list of agents, using the *print()* method in the Agent class.
5. Add a *getRandomAgent()* method to the Simulation class that returns the randomly selected agent (not an index, but the actual object) (see Lab 4).
6. Add a *communicate()* method to the Agent class that has as input parameter another agent (e.g. *communicate(self, other)*) and that updates the value of x accordingly (see Lab 4 for a detailed description). Use a separate method *calculateMu(self, other)* that calculates μ, assuming $m = 1$.
7. Add a *run()* method to the Simulation class that has as input parameter *times* the number of iterations to run the simulation. Write a loop inside this method that runs the simulation, and each time randomly selects two agents and has them communicate with each other.
8. Carefully read the Chen and Lan (2021) paper and make notes to summarise the simulation model.