In this exercise, you are supposed to implement a simulation of catching a sheep by a wolf.

The simulation includes two types of animals: a single wolf and a flock of sheep. Animals move over a meadow of infinite size. No obstacles are encountered there. The idea is simple — a wolf tries to hunt sheep and eat afterward, and sheep flee — though a bit clumsily.

According to what was stated above, a meadow is thought of as a Cartesian coordinate system — a 2D space with the center at the point (0.0, 0.0).

The location of each animal in a meadow is described by two floating-point numbers — coordinates. They can be both positive and negative.

At the beginning of the simulation, the location of all animals is established.

* 1. For sheep, it is randomly chosen from the uniform distribution from the range [-init\_pos\_limit, init\_pos\_limit].
  2. For a wolf, in turn, it is the center of a meadow — the point (0.0, 0.0).

The simulation runs in rounds according to the following order:

* 1. Each alive sheep moves:
     1. the direction is selected randomly (north, south, west, or east),
     2. the move distance in the opted direction equals sheep\_move\_dist
  2. A wolf moves:
     1. It finds the closest sheep
     2. It checks whether it is his range (distance shorter than wolf\_move\_dist).
        + - if so, a sheep is eaten — a sheep disappears and a wolf takes its location
          - otherwise, a wolf chases the selected (nearest) sheep, that is moves towards it by wolf\_move\_dist
  3. If more than one sheep is within the wolf's range, just one is being chased.

The simulation ends as soon as all sheep are eaten or the maximum number of rounds will be reached.

**Specific requirements for grade 3:**

1. To implement a simulation compliant with the aforementioned description, taking the following parameters' values:
   * maximum number of rounds: 50;
   * the size of a flock of sheep: 15;
   * init\_pos\_limit: 10.0;
   * sheep\_move\_dist: 0.5;
   * wolf\_move\_dist: 1.0.

To get random numbers, use the built-in *random* package or *NumPy*.

1. To implement displaying the basic information about the simulation state just after a round finishes. The following pieces of information should be considered::
   * the number of a round;
   * a wolf's position (with accuracy up to the 3rd decimal place);
   * the number of alive sheep;
   * if a wolf chases any sheep — which one it is (which is its ID or ordinal number used during initialization);
   * if any sheep was eaten — display information on which one.

Displaying the above pieces of information should not interrupt the simulation. No interaction with a user is assumed.

1. Utilizing *json* package, implement saving locations of each animal for each round, to the file pos.json. The content should be a list of dictionaries each of which corresponds to a single round of the simulation and it contains the following data:
   * 'round\_no' - the number of a round (integer number);
   * 'wolf\_pos' - a wolf's location (a pair of floating-point numbers);
   * 'sheep\_pos' - locations of all sheep (a list containing a pair of floating-point numbers for each alive sheep, or None/null for each sheep that has been eaten).

The JSON file should be well-formated. The proper indentation should be added. If a file pos.json already exists, it should be overwritten.

1. Making use of the *csv* package, implement saving the number of living sheep in each round to the alive.csv file. It should have two columns:
   * a round number (integer value);
   * the number of alive sheep (integer value).

Each row in the file should correspond to a single round. If the file alive.csv already exists — overwrite it.

**Specific requirements for grade 4:**

1. All requirements for the lower grade should be satisfied.
2. Use the *argparse* package to handle command-line arguments:
   * -c/--config FILE — an auxiliary configuration file, where FILE states for the name of a file;
   * -d/--dir DIR — a subdirectory where files  pos.json, alive.csv, and  — optionally — chase.log should be placed, where DIR means the name of a subdirectory;
   * -h/--help — the help;
   * -l/--log LEVEL — if events should be logged, where  LEVEL is a logging level (*DEBUG*, *INFO*, *WARNING*, *ERROR,*or *CRITICAL*);
   * -r/--rounds NUM — the number of rounds, where NUM is an integer number;
   * -s/--sheep NUM — the number of sheep in a flock, where NUM is an integer value;
   * -w/--wait — if simulation should be paused at the end of each round after displaying the basic information about the state of a simulation; to continue, any key should be pressed by a user.

All arguments should be optional and should be processed according to the following rules.

* + The option -c/--config indicates a configuration file, where information about  init\_pos\_limit, sheep\_move\_dist, and wolf\_move\_dist are saved.  The format is described below. If the configuration file was not passed, the default values for init\_pos\_limit, sheep\_move\_dist, and wolf\_move\_dist should be used (according to the requirements for grade 3).
  + The option -d/--dir indicates a subdirectory to save output files. If a requested directory does not exist, it should be created. If -d/--dir option was not passed, all files should be stored in the current directory.
  + The option -h/--help shows the manual of the program. It should contain a short simulation description and information about its parameters. Using this option results in closing the simulation just after showing the manual  — a simulation is not run.
  + The option -l/--log defines the logging level. If this option is passed, in the current directory or a subdirectory indicated by the -d/--dir parameter, a file chase.log is created. The file should be overwritten if it exists. If the -l/--log option is not passed, chase.log file is not created and no events are logged.
  + The option -r/--rounds is the maximum number of rounds in a simulation.
  + The option -s/--sheep indicates the number of sheep in a flock.
  + The option -w/--wait is a flag that indicates that a simulation should be paused after each round until a user presses any key. If this option is not passed, a simulation should not be interrupted.

All arguments should be properly validated (e.g. the number of rounds is an integer number greater than zero). If validation fails, a user should be informed about the error through an appropriate exception.

1. By using the *configparser* package, implement loading values for  init\_pos\_limit, sheep\_move\_dist, and wolf\_move\_dist from a configuration file defined in the argument -c/--config. A configuration file should be in INI format according to the below structure:
2. [Terrain]
3. InitPosLimit = 10.0
4. [Movement]
5. SheepMoveDist = 0.5

WolfMoveDist = 1.0

All values should be validated (e.g. length by which animals move is positive). In case of failing the validation process, a user should be informed about it through an appropriate exception.

1. Using the *logging* package, implement logging events to a file with the name chase.log. All events at the logging level specified by -l/--log or higher, should be logged:
   * for the level *DEBUG* (10) all function calls should be logged as well as values returned by these functions (if any);
   * for the level *INFO* (20) all activities undertaken by a program should be logged (e.g. setting the initial positions of animals, an animal move, etc.);
   * for the levels *WARNING* (30), *ERROR* (40), and *CRITICAL* (50) the proper events should be logged.

The logging level *DEBUG* is intended for developers to track possible bugs in a program. In turn, the *INFO* level is intended for users and aims at storing information on the course of a simulation so that it could be reconstructed in the future. For higher levels, like *WARNING*, *ERROR*, and *CRITICAL*, events being logged depend on the way a program was implemented. If no events seem appropriate to be logged at these levels, it is not required to log anything.

**Specific requirements for grade 5:**

1. All requirements for the lower grades should be satisfied.
2. Use package *distutils* or *setuptools*in order to create a publishable package *chase —*installable in the virtual environment. The code should be well organized so that the logic could be reused. The piece of code responsible for the simulation run in the console mode should be located in the module \_\_main\_\_.py inside the package. Having installed the package, a simulation will be run with the command python -m chase [ARG], where ARG are the optional arguments of a simulation (see: requirements for grade 4).
3. During the project defense, you should show how to create a virtual environment and how to install the newly created package *chase.*Then, you should run a simulation from the installed package