General constraints for code submissions Please adhere to these rules to make our and your life easier! We will deduct points if your solution does not fulfill the following:

- If not stated otherwise, we will use exclusively Python 3.6.
- If not stated otherwise, we expect a Python script, which we will invoke exactly as stated on the exercise sheet.
- Your solution exactly returns the required output (neither less nor more) – you can implement a `--verbose` option to increase the verbosity level for developing.
- Add comments and docstrings, so we can understand your solution.
- (If applicable) The `README` describes how to install requirements or provides addition information.
- (If applicable) Add required additional packages to `requirements.txt`. Explain in your `README` what this package does, why you use that package and provide a link to it's documentation or GitHub page.
- (If applicable) All prepared unittests have to pass.
- (If applicable) You can (and sometimes have to) reuse code from previous exercises.

From the lecture you have learned everything about Multi-criteria Optimization. As you have already implemented EAs and BO you are tasked with implementing various small methods you would need to use those implementations for multi-objective optimization.

1. **A-priori procedures** [2 points]
   Complete the code in `src/apriori` for the presented a-priori methods from the lecture for determining an optimal point with multiple-criteria and pass the corresponding unittests.
   (a) Implement the weighted total. [1pt.]
   Your implementation should satisfy the test in `test_a_priori.py::TestAPrioriWeightedSum`.
   (b) Implement the lexicographic procedure. [1pt.]
   Your implementation should satisfy the test in `test_a_priori.py::TestLexicographical`.

2. **Pareto front** [3 points]
   In this exercise we'll be using the Wine classification dataset. We will use a simplified version where the criteria you optimize for acidity and nonflavanoids (ash in the 3D example). Complete the method `pareto` in `src.pareto` to determine the pareto front on this data. For simplicity sake we always minimize, i.e. your objective is to find the wine with the lowest acidity and nonflavanoids.

3. **Non-Dominated Sorting** [3 points]
   Implement the non-dominated sorting mechanism as presented in the lecture (`NDS` in `src.pareto`). Your implementation should satisfy the test in `test_paret_front.py::TestNDS`.

4. **Crowding Distance** [3 points]
   Implement the crowding distance method. For this exercise you are given a front of points for which you have to compute the distance to it's neighbors (`crowdingDist` in `src.pareto`). Your implementation should satisfy the test in `test_paret_front.py::TestCD`.

5. **Compute the Hypervolume** [3 points]
   Compute the hypervolume for a given front and a corresponding reference point. For ease of implementation you will only have to implement a method that computes the hypervolume in 2D, i.e., the area of a polygon[1], see `computeHV2D` in `src.pareto`. Your implementation should satisfy the test in `test_paret_front.py::TestHV`.

[1] `https://en.wikipedia.org/wiki/Shoelace_formula`