General hints for code submissions To make it easier for us and others to understand your solutions please follow these guidelines:

- If available use the template file to create your solution.
- Please add comments so we can understand your solution.
- Please make sure to load all required packages at the beginning of your code.
- Only use relative file paths for `source()`, `load()`, etc.
- Each exercise directory contains a `skeleton` folder where preliminary `R` files are located.
- Use these `R` files as a basis for creating your solution which should be contained in a main `R` file named as the content of the exercise, e.g., `evaluation.R`.
- You can (and sometimes have to) reuse code from previous exercises.
- The points indicate the difficulty of the task.
- If not stated otherwise, we will use exclusively R 4.0 or greater.

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 for the presented a-priori methods from the lecture for determining an optimal point with multiple-criteria.
   (a) Implement the weighted total. Find the optimal wines with different trade-offs (determined by the weights) between acidity (malic), ash and flavanoids. Always weight ash with 10%. [1.pt]
   (b) Implement the lexicographic procedure. [1.pt]

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 function `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` function).

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 its neighbors (`crowdingDist` function).

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] (`computeHV2D` function).

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