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 about evolutionary algorithms and simple methods of hyperparameter optimization.

1. **Evolutionary Algorithms** [11 points]
   The lecture taught you all about the individual parts of a simple evolutionary Algorithm. Your task is to implement mutation, recombination and selection mechanisms. Specifically:
   (a) Implement `Uniform` and `Gaussian` mutation mechanisms (in the member class). Your implementation should pass all tests in `test_mutation` [3pt.]
   (b) Implement `Uniform crossover` and `Intermediate` recombination mechanisms (in the member class). Your implementation should pass all tests in `test_recombination` [3pt.]
   (c) Implement `Neutral`, `Fitness-proportional` and `Tournament` parent selection mechanisms (in the EA class). Your implementation should pass all tests in `test_selection` [4pt.]
   (d) In the step method of the EA class, create one offspring per selected parent either through recombination or mutation. You can use the frac_mutants parameter to trade-off both approaches. [1pt.]

2. **Basic HPO** [3 points]
   Implement a simple HPO method (either grid or random search) to optimize three parameters of your EA. Specifically your method has tox handle three categorical parameters (mutation, selection and recombination). When evaluating your method (see `test_hyperparameter_search`) we will overfit to the Ackley function you have seen in the lecture.

3. **Code Style** [1 point]
   On every exercise sheet we will also make use of `pycodestyle`[1] to adhere to a common python standard. Your code will be automatically evaluated on every push and you will be informed if the test fails. To check it locally, first run `pip install pycodestyle` and then run `pycodestyle --max-line-length=120 src/` to check your source file folder. Alternatively run `make checkstyle`

[1]former pep8