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 3.5.
- 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.

After you now know how to use algorithm configuration, your next task is to configure the SAT solver *SATenstein* to optimize its performance using SMAC.

The scenario consists of the following directories

- `SATenstein`: a directory with the binary of *SATenstein* and a basic framework for an algorithm wrapper that uses `genericWrapper4AC`[1] (to install the wrapper, clone the repo and run `python setup.py install`)
- `indu`: a directory with industrial instances from various SAT competitions. All instances are satisfiable!

1. **Configuration of *SATenstein*** [10 points]
   Given the above mentioned files, your task is to optimize the performance of *SATenstein* on the provided instances with *SMACv3*[2]. You will have to clone the SMAC repository to use its command line interface[3]. You can however choose to use SMAC as a python package instead[4]. To use either method of running SMAC, please work through the following steps:

   - Split the instances in a training (`training.txt`) and test set (`test.txt`) – write a bash or Python script to do so and upload it into your `src` folder. The instances should be split evenly into training and test sets. The given instances are **heterogeneous** and consist of instances from two domains. You have to be mindful how you split the instances. The instance names indicate if they are from the same domain.
   - Complete the Python script `wrapper.py`; the call to *SATenstein* should look like:
     `satenstein/ubcsat -param1 value1 ... -paramN valueN -seed <int> -inst <instance> -target 0 -r satcomp -cutoff <max search steps> -timeout <max running time>`
     e.g. `satenstein/ubcsat -alg satenstein -adaptive 0 -inst indu/factor-3023-3607.cnf -target 0 -seed 0 -r satcomp -cutoff -1 -timeout 5`
     The cutoff parameter is used to limit the number of search steps whereas the timeout parameter limits the runtime! So you have to set cutoff to -1 for unlimited search steps.

     Verify the functionality of the completed wrapper with the following call (in the scenario folder):
     `python satenstein/wrapper.py indu/factor-3023-3607.cnf 0 5 0 1 -adaptive 1`
     This calls the wrapper to run SATenstein on instance `indu/factor-3023-3607.cnf`, with 0 as `instance specifics`, a `cutoff` of 5 seconds and a `runlength` of 0 (which should be ignored by your wrapper and instead set to -1), `seed` 1 and the parameter `adaptive` set to 1.

[1] `https://github.com/mlindauer/GenericWrapper4AC`
[2] `https://github.com/automl/SMAC3`
[3] `https://automl.github.io/SMAC3/stable/quickstart.html#command-line`
[4] `https://github.com/automl/ParameterImportance/blob/master/notebooks/interface_example.ipynb` (see lines 2-6)

The final printed line will look something like this:
`Result for ParamILS: <Status>, <runningtime>, <runlenght>, <quality>, <seed>`
All the paramters for the call to the wrapper will be automatically set by SMAC during optimization, so your wrapper only needs to construct the cmdline call to SATenstein.

- Complete the scenario file (`scenario.txt`) that defines the following characteristics of the configuration scenario
  - use your generated training and test files (`train.txt` and `test.txt`)
  - the algorithm is non-deterministic
  - optimize runtime
  - the overall objective is the mean10 (PAR10) score
  - the cutoff time time will be 5 seconds
  - the configuration budget will be 900 seconds
- Run *SMAC* with the above defined scenario, report the hardware you used and the performance of the configured *SATenstein* on the test instances.
- Validate the default configuration of *SATenstein* on the test instances – you can use SMACs validation script and the script provided in `src/read_validation_data.py`. This will print some details about all validated configurations. Report the performance of the default configuration as well as the incumbent.
- Upload all files you created, i.e. training and test files, SMAC output as well as the validated runhistory.

2. **Feedback** [Bonus: 0.5 points]

For each question in this assignment, state:

- How long you worked on it.
- What you learned.
- Anything you would improve in this question if you were teaching the course.

This assignment is due on 19.07.19 (10:00). Submit your solution for the tasks by uploading a PDF to your groups BitBucket repository. The PDF has to include the name of the submitter(s).