# INDU—Individual Assignment in Programming

February 21, 2022

*Developed by Lars Arvestad for the course DA2004 - Programmeringsteknik för matematiker. Modified by Kristoffer Sahlin and Anders Mörtberg.*

## Contents



## General requirements

Basic rules for all project submissions:

- Work individually, i.e., write your own code and find your own solutions.
- Report all help and guidance you have received.
- Program in Python 3.
- A valid solution works without error messages on valid input.

After submission, there will be a review phase when you will review peers' solutions, just as we did with the labs. You cannot pass INDU if you have not reviewed three other submissions.

To help with the programming, you can use all modules in Python's standard distribution, as well as `matplotlib`, `numpy`, and `scipy`. If there are additional modules you think could be useful, you should ask a teacher first.

## Submission

You must submit your code together with a short report on PeerGrade.

The report must be a PDF (`.pdf` file format) to ensure that everyone can read it. The code must be easy to test. Be sure to include any relevant data and test files with your submission.

## Report

Your report should not be long, just useful. **Maximum length is 3 pages.** You can choose which font, font size, margin size, etc., you want as long as the report is readable.

The report **must** include:

1. Which program and task(s) you have implemented.

2. How the program is started and used.
3. Which libraries/modules are used and how these are downloaded and installed if they are not part of Python's standard distribution.
4. How you structured your program (which files contain what, etc.).

The report **may** also contain reflections on:

- code design,
- which algorithms are used and why,
- which data structures are used and why,
- time and memory efficiency of the code,
- ...

**Note:** the main purpose of the report is to make it easier for those who will use and test your code! If you do not have a report that contains the four mandatory points above, your project will be rejected.

# Grading

To obtain grade E, we require that your program solves the specified task and meets general (see above) and project-specific requirements (including a report).

For higher grades, you must meet our additional quality criteria. The project and how much of it you implement also affects your grade. Some projects can only give you grades E-C, while others can give you grades E-A, depending on difficulty.

The three criteria that raise your grade are:

1. Good code.
2. Good error handling.
3. Testing.

These criteria will be scored with 0-2 points according to:

0. Does not meet the criterion.
1. Meets the criterion.
2. Meets the criterion well.

The grades and project goals are then:

A. Very good and clear code. Very good error handling and testing. (6 points in total)
B. Very good and clear code. There is both error handling and testing. (4-5 points in total, of which 2 on criterion 1)
C. Very good and clear code. There is error handling or testing. (3 points in total, at least 1 on criterion 1)
D. The code is of good quality, possibly with error handling or testing. (2 points in total, at least 1 on criterion 1)
E. The code works and solves the task. (0-1 points in total or 0 points on criterion 1)

Below is a matrix that summarizes which grade you get on the project based on which project you have done and how many points (0-6) you have received on the grade-raising criteria:

| Poäng: | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Simpler project | E | E | D* | C* | C* | C* | C* |
| Harder project, task 1 | E | E | D* | C* | C* | C* | C* |
| Harder project, task 1 & 2 | E | E | D* | C* | B** | B** | A |

C*/D*: At least 1 point on criterion 1.
B**: At least 2 points on criterion 1.

Below follows a clarification of our grade-raising critera.

## Criterion 1: Good code

To meet this criterion, the code must follow the instructions in "Basic principles of programming" and the section on "Good code" in the course notes. This includes:

- Well-chosen and informative names of identifiers.
- Appropriate amount of informative comments explaining important steps and assumptions in the code.
- All functions, classes and methods must have a documentation string that explains what their purpose and function is in addition to what is already stated in the chosen identifier.
- Suitable line length (preferably less than 80 characters, definitely less than 100).
- Appropriate length of functions and methods.
- Clear file structure and organization of code.
- Good partitioning of code into functions and classes.
- Clear separation of functions/methods that perform calculations and those that perform user interaction. This means you should have special functions or methods for calculations that do not contain statements such as `print`, `open`, or `input`.
- Functions should not be dependent on global variables unless necessary. Use of global variables requires a justification in comments.

## Criterion 2: Good error handling

The program must not crash for certain runs, neither for random effects that occur in simulations nor due to incorrect parameters or inappropriate input from the user. Use appropriate error handling through `try-except` blocks to avoid crashes. You should also raise appropriate exceptions using `raise` when necessary.

In order to meet this criterion well, error handling must be handled in a good way as described in the course notes. For example, you should not lift an exception just to capture it directly afterward of use `try-except` when it would be more appropriate to use an `if`-statement.

## Criterion 3: Testing

Why should the reviewer trust that the program works? Claiming that "it seems to work" in your report is not enough. You must justify for the reviewer why one can trust that your program works. Unit tests are required to meet this criterion well (by using the `unittest` module or comparable alternative).

In order to write good tests, functions and methods must be written in a way that makes it possible to test them in a meaningful way. If you have difficulty writing good tests, it is a sign that you should rethink the design of your program. A good rule of thumb is that by writing short functions that *return* a result, you can much more easily write tests that verify that everything works as intended.