INDU—Individual Assignment in Programming

December 5, 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.
- Make a note of any help and guidance you have received.
- Program in Python 3.
- Your solution should work without error messages on valid input.

After submission, there will be a review phase where you 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 any module 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 should be easy to test. Be sure to include any relevant data and test files with your submission.

Report

Your report should not be long. **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. A description of 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 (1.–4.), your project will be rejected.

Grading

To pass, your program must solve the specified task and meet the general (see above) and project-specific requirements. You must include a report as described above.

For higher grades, you must meet our additional quality criteria. 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 (3 subcriteria, 2p each, so 6p total).
2. Good error handling (2p).
3. Testing (2p).

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.

Finally, any bonus points (0, 1, or 2) from the labs will be added.
Your grade for the project is determined by your choice of project how many points you receive (0-12) and is calculated using the following matrix:

Points:	0-2	3-4	5-6	7-8	9–
Simpler project	E	D	C	C	C
Harder project, task 1	E	D	C	C	C
Harder project, tasks 1 & 2	E	D	C	B	A

Below follows a explanation 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. Your code should also be organized appropriately into classes, i.e., should be written in an object-oriented style.

The points for this criterion are divided into 3 subcriteria:

A. Documentation (2p)

- An appropriate amount of informative comments that explain the key steps and assumptions in your code.
- Each function, class, and method should have a documentation string that explains its purpose (beyond what is clear from its identifier).

B. Readability (2p)

- Well-chosen and informative names for identifiers.
- Appropriate line lengths (ideally under 80 characters, definitely under 100).
- Appropriate length of functions and methods (half a laptop screen max).

C. Structure and object-orientation (2p)

- Appropriate division of code into classes. To satisfy this criterion well, your code **must** be organized in an object-oriented way. The majority of the code should be within suitable classes, with only a few top-level functions (such as a `main` function).
- Good organization of code into methods/functions so that code duplication is avoided.
- Clear file structure and organization of code.
- Clear separation between methods/functions that do calculations and those that do user interaction. This means your methods/functions for calculations should not contain calls to `print`, `open`, `input`, etc.
- Functions should not depend on global variables, but you can use global constants (that is, variables that never change value while the program is running) if appropriate. If you use any global constants, you should motivate them in your comments.

Criterion 2: Good error handling

The program must not crash on certain runs, neither as a result of random effects that occur in simulations nor because of incorrect parameters or malformed input (either from the user or from a file). Implement appropriate error handling using `try-except` blocks to avoid crashes. You should also raise appropriate exceptions using `raise` when necessary.

In order to meet this criterion well, you should use error handling in a reasonable way as described in the course notes. For example, you should not lift an exception just to capture it directly afterward or 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).

To fulfill this criterion, you must first write your functions and methods in a way that makes it possible to meaningfully test them. 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 verifying that everything works as intended.