# Final Exam: Project Guidelines

## 1. Deliverables

The **requirements** are minimal and flexible: at least one Jupyter notebook (.ipynb) file containing your analysis in English, math, and Python code.

It's up to you to decide the format or approach you're going to take. Several well-known approaches we've seen so far (which you may mix and match) are:

- **Technical report**
  You describe your problem. Then you lay out your approach to it, including any assumptions, and present your results. You'll need to pay special attention to your methods and implementation details. Basically, "I had to do X, here's how I did it".
- **Tutorial**
  Like a technical report, focusing more on the "how-to" steps; and delving into more details about how things work. This is very useful for people who are new; or even as a way to explain a new concept to yourself before teaching it to others.
- **Scientific article**
  A well-known, formal, structure, like the articles you may see in a scientific journal or conference. The code may be "on the side", separate from the article itself, although this is not ideal.
- **Popular science article**
  The less formal cousin of the scientific article. You're not so constrained by structure or formality requirements; and you're free to use your own imagination and style. You'll see these in sites like Medium, or in some YouTube video essays. These are very suitable for explainers and reviews.
- **Multimedia product / explainer**
  A variant of a popular science article is a multimedia product. In addition to your explanations, math, and code in the notebook, you create a different type of content: preferably one which includes interactivity – like an interactive data visualization, or algorithm exploration. Don't forget that the primary objective here is the math, not the product itself.
- **Code repo / library / framework / app**
  Self-explanatory. It should be suitable as an additional deliverable to the notebook when creating a reusable project (or one which has reusable elements). Some examples may include automation scripts, or custom libraries to solve a problem.

Once again, feel free to mix and match, and add your own ideas.

## 2. Grading principles

The following point will specify technical guidelines for evaluating your project. Since those were made to be abstract (in order to cover many cases), here are the main principles that we're looking for. These apply not only to this course, but also to the industry and academia.

1. A good, well-formed **problem formulation**. Well-known problem significance. Pros and cons of similar solutions (if such exist). Known constraints (explicit and implicit) and assumptions. In addition, the translation between the "real-world" problem and its mathematical counterpart is correct.

2. Deep **mathematical understanding** of the topic(s) at hand, and ability to comment on adjacent topics (like evaluating the results, relaxing some constraints, writing better math expressions). Creativity and rigor (to whatever grade is possible) in applying the math concepts to the real-world problem.
3. **Code and presentation quality**. The code is readable, easy to maintain if needed, and well-documented. It follows all principles of quality code in Python (or another chosen language, if necessary). Appropriate use of external libraries and tools. The text and math expressions are correct, with no (or very few) errors in grammar or spelling; they're well-written and explain the task, steps, approaches, etc., well.
4. A variety of **methods and applications**. Well-known previous approaches. Demonstrated correct and clear usage of different methods, techniques (e.g., for data analysis, or optimization), and approaches related to the problem. If applicable, a variety of experiments for the task at hand.
5. **Communication**. Clear, easily understandable deliverables, such as project structure, intent, documentation, etc.

## 3. Minimum criteria

Projects which fail one or more of the following criteria will be given a failing grade.

1. **Adherence to legal requirements**
   The project should comply with the law in Bulgaria by the date of the exam. The code should not intend or attempt to break the examiner's (or someone else's machine), even if it relates to security research.
2. **Functional code**
   The code works with no errors (or very small, easy to fix errors)
3. **Deliverables**
   All deliverables (at minimum, a Jupyter notebook) are available and submitted within the assigned term. No deadline extensions are allowed
4. **Adherence to project guidelines**
   The logistics (project guidelines, deadlines, deliverables, etc.) and documentation (exam participation survey, self-assessment, etc.) are carefully followed. The project is of a reasonable difficult, meeting the expectations of careful and reasonable preparation.
5. **Academic integrity**
   No cheating or plagiarism is allowed. Sources are documented and credited.

## 4. Grading criteria (short)

Don't be scared of the long list of criteria. It exists merely to handle many cases and make grading easier and more consistent.

1. **Problem understanding, formulation, and significance (0 – 15)**
   You know what task you're solving; and you know why it's important or at least interesting.
2. **Writing layout (0 – 20)**
   You can clearly express your thoughts in a well-structured, well-formatted, easy to understand way.
3. **Mathematical understanding and research (0 – 15)**
   You know the math and ideas behind your problem, and possibly some other stuff. This is the theoretical part of project.
4. **Code quality (0 – 15)**

You write clean, well-formatted, well-documented, and easy to understand code. You know what refactoring and testing is. You're proud of the code you've written.

5. **Methods; data handling / analysis / processing (0 – 20)**
   You can apply what you've already learned in code, using the correct methods; you perform checks and tests, to ensure quality.
6. **Communication (0 – 15)**
   You express all of the above in a good way, well suited to your audience.

## 5. Grading criteria (full)

Keep in mind that while these have associated point values, they are flexible. We're using the point values only as guidelines to ensure that your self-assessment and the examiner's assessment have a common ground. Some projects may not fit in these criteria as much as others. The previous point outlines the principles of a high-quality project in a better way.

1. **Problem understanding, formulation, and significance (0 – 15)**
   - The problem is well and clearly formulated
   - The significance of the problem / approach / method, etc. is well demonstrated
   - The project scope and objectives are clear
2. **Writing layout (0 – 20)**
   - All written deliverables are well documented, with a special emphasis on the methods and / or experiments. The document(s) is / are well formatted
   - The text is well structured into sections
   - All explanations are clear and well written. The documentation is complete and easy to understand
3. **Mathematical understanding and research (0 – 15)**
   - The relevant mathematical concepts, ideas, formulas, expressions, etc. are well explained and clearly laid out
   - The theory behind the project is well known and understandable, as needed
   - If applicable, adjacent / similar concepts are also laid out for comparison
   - If applicable, proofs or other supplementary materials are provided as references or in the deliverables
   - References to all sources (documents, articles, codebases, etc.) are well-maintained
4. **Code quality (0 – 15)**
   - The code is readable, clean and easy to maintain
   - All coding best practices are taken seriously. For example, there is a clear organization and separation of concerns; the code is well-documented
   - The appropriate data structures, algorithms, and libraries are used, as per the project requirements
   - The code is tested (as needed, unit tests are only one example; and they're not always necessary)
   - If applicable, the code is efficient. In addition, its efficiency may be analyzed in depth
5. **Methods; data handling / analysis / processing (0 – 20)**
   - The project will likely involve data usage in some way, even if it's not a standard dataset. Data may be gathered (via experiments), generated, simulated, etc. If not, the methods / approaches to the solution will be graded instead
   - All methods are correct both in terms of idea and implementation. They are validated, and their scope / limitations / assumptions are well laid-out

- Any methods / algorithms are performed and tested thoroughly, following the best practices
- If applicable, the correctness and assumptions of all methods are well tested and documented.
- Any data processing or analysis is done correctly, following the best practices; including cleaning, visualization, metrics, etc.

6. **Communication (0 – 15)**
   - This is similar to the layout, but applies to all deliverables and the project as a whole
   - There is a clear conclusion (and possibly, outlines of next steps) with respect to the problem formulation. Of course, the conclusion does not need to be positive.
   - The style of writing / code is appropriate to the selected audience; with the desired language specifics (e.g., use or avoidance of technical terms, as necessary)
   - The additional deliverables (if applicable, e.g., multimedia, apps, libraries, interactive elements) are well-presented
   - Integrity (both in writing and project implementation) and lawfulness are taken seriously