General instructions:

- For this project, we are labeling how understandable a code file is. Since we are looking at code files, most of them should be written in a programming language, thus a formal language with unambiguous instructions and grammar.
- However, having an unambiguous language doesn't mean it is easy to understand, and because programming languages generally have very different syntax and grammar from natural languages, a well-written and functionally-correct code can be very hard to understand and maintain.
- Thus we don't emphasize the codes themselves. Rather, we look at the comments or comments-like strings that are 1): optional and 2): that do not influence the actual functionalities of the program, including:
    - Comments/function docstring
    - Type hints/Variable annotations
    - Meta-info like file structure, paper references, API references, links to more detailed documentation, etc.
    - Coding style, including variable and function names, and whether they are explanatory of their functions
    - If applicable, strings that are helpful in certain ways, like Raise error string or print statements with status info.
- The order of the listed elements is not important, though we agreed that comments and function docstring are the most helpful ones, and the rest can be a bit subjective.
    - A more experienced programmer can infer more things from variable definitions and type hints alone, for example if they follow some conventions or if one is familiar with a certain type of algorithm, though it might not be as obvious to a less experienced programmer. We decided that we should score the code files based on comments, function docstring, and other elements that are written in plain English. The other elements are complementary <u>and should only be considered if the code file is worth a score of at least 2</u>. We will explain the numbering system in the following section and what they represent
- Note that we don't check for correctness, because it is impossible and unnecessary for this task.

Criteria:

- We score the file based on how much of the file is <u>properly explained</u>. Here is how we decide if a segment is properly explained:
    - Functions/class methods:
        - They are properly explained if they have corresponding doctrines in their definitions. The quality of docstring may vary depending on the context, but a basic docstring is required for any function that expects some arguments or returns something, and the docstring should clearly explain what the purpose of the function/method is
    - Variables:
        - This is more important if variables are poorly named. What counts as poorly is subjective, but in general we are looking for clear names that

        represent the underlying objects to some extents. Not saying all variable definitions need comments, but if there are many and it is getting confusing, we are looking for comments at that point.
    - Some files we encounter are configuration files, meaning the majority of the file consists of variable creation and assignment. We treat them as normal variables and apply the same rules,
  - Logic blocks:
    - By logic blocks, we mean standalone segments of codes that are not part of any function but serve some purposes, like for loops or a series of API calls in the global scope.
    - In a perfect world, everything will be properly wrapped in functions, but in reality we just treat it as a function without a function definition, so we need to understand the purpose of the logic blocks.
    - It generally requires more comments than the docstring of a function of the same length, mainly because it lacks the modularity provided by functions/methods. The amount required varies depending on the actual contents of the logic blocks, but in general we treat them like functions
- 2 edge cases:
  - 1): Meta-information while the code file itself is poorly annotated. We focus mostly on comments and docstrings, but we have run into files where the codes are not very well annotated, but there are links to more comprehensive documentation, or explanation of the file structure at the beginning, while the contents of the file remain confusing. We decided that this is NOT sufficient, and therefore should only be considered in a supplementary fashion. Thus, <u>they should only be considered if the code file is worth a score of at least 2</u>
    - Note this is different from linking an API reference above an API call. This type of reference clearly relates to a specific line, thus making it easy to follow or even ideal. The edge case discussed here refers to the case where the author puts an URL at the beginning of file, say a link to Pytorch documentation, without any further explanation.
  - 2): auto-generated files. We limited our data to python files, though we still run into some auto-generated files, and being auto-generated means there is little documentation in them. We decided that we treat them the same as all other files, the logic being there is no inherent difference between human-written code and computer-generated code, and it is almost impossible to be 100% sure if a code is auto-generated just by looking at the code alone. The rule we employed is that if one can tell a file is auto-generated, for instance, from a comment at the beginning, “this file is generated by …”, it should be assigned a score of 1, even if it has nothing else commented. We will explain the scoring system in the following section

The scoring system:

- **0: No comments/documentation**

    - There are little to no useful comments explaining the code, making the code very hard to maintain.
    - <u>There is NO useful documentation anywhere in the code</u>, meaning there no docstring for function, method or class definition, nor comments throughout the code in general.
    - <u>If there is no function, method or class definition in this file</u>, 0 means there is no explanation of any part of the codes, including comments or annotations
    - <u>Notes</u>:
        - A random one-line comment that explains a self-explanatory line (like a read/write command) doesn't count as helpful info. It must provide more context than the code itself.
        - Commented out codes do not count as comments.
        - The organization/functionality of the code itself is irrelevant for this category. Even if it is a standard configuration file with minimal need for comments/clarifications.
    - <u>Bottomline</u>: 0 means there is nothing.

- **1: Some comments but not enough**
    - There are some comments/documentation scattered throughout the code, but the vast majority of the document is still unclear or confusing.
    - Quantitatively, it means less than 50% of the code is properly explained, but there are some helpful comments.
    - <u>Bottomline</u>: a 1 can be anything that is marginally better than nothing. Overall if more than half of the code seems to be missing documentation, but there are something, it is probably a 1.

- **2: More comments with better consistency**,
    - At least half of the document is clear about what its purpose is, but there are still points of confusion/unlabeled code segment that is unclear.
    - Quantitatively, at most 50% of the document still seems confusing, and the reader should has a general idea of what the code file is aiming to do.
    - <u>Bottomline:</u> 2 is more comprehensive than 1, but not perfect

- **3: Great effort with comprehensive documentation**
    - Almost all of the code file should be clearly documented
    - <u>Must have</u>:
        - Well formatted docstring for all class and function definition in this file.
        - Consistent variable definitions.
    - Code logics in the file needs to be clear with possible points of confusions explained.
    - <u>Note</u>:
        - We are not looking for every single line of code to be annotated/commented. If a for loop is clear of what it is doing by using

        good variable names, or a function is really self-explanatory, it is fine to go without comments, but we expect these cases to be rare. For example:
        - Function that starts with “test_*” is clear that it is a unittest, but we want to know what it is testing, so simply recognizing that it is a unittest is not enough
        - A simple For loop consists of:
            - *for weekday in days:*
              Income = find_revenue(weekday)
              ….

          This can be considered clear without further explanation, however, another more complicated for loop needs to have proper explanations, even with proper naming conventions
    - <u>Bottomline:</u>
        - 3 means nearly all of the code file should be properly explained, thus having very few points of confusion.

- **4: Comprehensive annotations + additional helpful information**,
    - In addition to having nearly perfect annotation, there are additional optional comments/annotations, like <u>example usages, overall file explanations, API/paper references, or any other information that is helpful but are not docstring/comments/type hints</u>
    - <u>Exception</u>: License information is excluded as it by itself doesn’t really improve the readability of the code and provides no additional resources for understanding.
    - <u>Notes</u>:
        - Example usages can be interchangeable with proper docstring, and some might prefer this. So if a function only has sample usages but no documentation, the annotator decides if the usages are good enough to be warranted a 4
            - Only documentation + no usage examples = at most 3
            - No documentation + usage examples = can be 4
            - Documentation + usage examples = 4
        - The distinction between a 3 and a 4 can be small. They are separated because we feel having this additional information is definitely more helpful than simply having the documentation, and only a small number of code files we look at actually include these.
            - For example, if a file is properly annotated, including one extra line at the top “This file includes the unit tests for the xxx library”, should this be a 3 or a 4? This becomes a subjective task, and both are acceptable in this case
    - <u>Bottomline:</u> 4 is a 3 with additional information.