# Assignment 2 Marking Criteria

Your program/code/software will be assessed against the following criteria:

* **Readability**: The ease with which your code/software can be understood by a human. Usually helped by: an abundance of clear, concise, informative source code comments; the use of naming conventions; a consistent and standard source code layout (achieved via standard and consistent code indentation, blank lines and the adherence of standardised documentation syntax); and producing appropriate auxiliary documentation as part of the submission.
* **Structure**: The degree to which the code has been organised into relevant blocks, files and other structures as appropriate.
* **Validity**: The severity and quantity of any logical or functional errors and the presence of appropriate tests and internal checks. Evidence of testing can be provided in documentation and not in source code.
* **Efficiency**: The extent to which the code reduces unnecessary computation and limits memory usage.
* **Functionality**: The overall functionality and usability of the software and any associated website.

Auxiliary documentation about the software is expected to include a simple [README](https://en.wikipedia.org/wiki/README) file, and other documents as appropriate providing further details about the software use and development. What is relevant to include as auxiliary documentation will depend to an extent on how well the software development has gone. It is expected to include a reference list of sources used to develop the software. It may also include user documentation to help users use the software. It might highlight potential uses of the software and it might outline ideas and intentions for further developing the software. It might document issues encountered during development and how these were overcome (or not). It might provide some details about the thought processes going into the software design, and outline the software development process followed.

The README should either be a [Markdown](https://en.wikipedia.org/wiki/Markdown) file or a simple [Text](https://en.wikipedia.org/wiki/Text_file) file that: either provide a contents (a simple list of what all the files are in the distribution) or links to a [Manifest](https://en.wikipedia.org/wiki/Manifest_file) file which essentially provides the content details; outline what the software is, how it can be run, and what is to be expected when it is run; and, declares the [Software Licence](https://en.wikipedia.org/wiki/Software_license).)

The need for documentation in addition to the README will depend on how detailed the README is. Evidence that the code has been tested as it has been developed is important. Some of this evidence might be in the source code files, but some details about testing might be included in a document about the development process which might detail any remaining issues as well as outline any issues that were encountered during development and how these were dealt with or remain unresolved.

Source code documentation/comments are expected to be detailed and extensive and ideally should make it clear that the developer knows what the code does. Each class, function and variable should be described. Standardised documentation syntax is expected for functions - listing positional arguments and detailing any returned values.

The code/software not compiling or running does not *necessarily* mean the work will be classed as a fail, especially if an earlier version of the code/software does work and completes some of the task set and an attempt has been made to document the process of resolving issues. The final mark may result from a combination of elements from across the marking scheme:

|  |  |
| --- | --- |
| **Grade** | **Typical Criteria** |
| **High Distinction** | **Readability**: The code/software will have excellent documentation. Ancillary documentation is likely to detail a professional [development process](https://en.wikipedia.org/wiki/Software_development_process). The software source code will contain appropriate useful comments that explain what the code does. The code will be laid out clearly and consistently adhering to a style guide. The program itself may come with structured help files which may be embedded in the software system.  **Structure**: Where appropriate, the code will follow standards of [loose coupling](https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29) and [high cohesion](https://en.wikipedia.org/wiki/Cohesion_%28computer_science%29), and may utilise professional [design patterns](https://en.wikipedia.org/wiki/Software_design_pattern). Associated Graphical User Interfaces and websites will be well structured, in a manner that potentially shows the influence of [Information Architecture](https://en.wikipedia.org/wiki/Information_architecture) literature. For websites this will include well designed structuring of associated directories and files, with an exemplary separation of functionality (HTML; CSS; JavaScript).  **Validity**: The code will show an appropriate level of testing and internal checks, both to ensure the program’s validity and to prevent fatal issues during running. There will be checks to help prevent the system exiting in error (especially as a result of users providing unusual input values or clicking buttons and pressing keys accidentally).  **Efficiency**: The code will be very efficient in terms of both memory and processing times. There may be evidence of profiling and timings in auxiliary documentation.  **Functionality**: The code will include the core functionality, enhanced with additional functionality if this is appropriate for the task set. The software will be designed to produce informative error messages that advise a user if an error is encountered. The system will be designed to prevent users making mistakes and/or that mistakes are resolved with little difficulty. Any associated Graphical User Interfaces or websites will be well presented and consider user accessibility issues. The write up should consider [Human-Computer Interaction](https://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction) and [usability](https://en.wikipedia.org/wiki/Usability) as appropriate.  Overall, the code/software will be user and developer friendly and of a release standard, or close to it. |
| **Distinction** | **Readability**: As for **High Distinction**.  **Structure**: Where appropriate, the code will show some thought into the development of structural units (like classes) that enhance the reusability and readability of code. Associated Graphical User Interfaces and websites will be thoughtfully structured. For websites this will include good structuring of associated directories and files, with separation of functionality (HTML; CSS; JavaScript).  **Validity**: The code will show an appropriate level of testing and internal checks, both to ensure the program’s validity and to prevent most if not all fatal issues during running.  **Efficiency**: The code will show that some thought has been put into efficiency and there will be some evidence that the code has been timed.  **Functionality**: The code will include the core functionality, perhaps enhanced with additional functionality although this may not be entirely successful or well reasoned. [Usability](https://en.wikipedia.org/wiki/Usability) should clearly have been a consideration in any associated Graphical User Interfaces and websites.  Overall, the code/software will be of a near professional release standard with only minor usability issues. |
| **Merit** | **Readability**: The code/software will have appropriate documentation, including ancillary documentation which may detail the thought processes behind the code functionality and development. The software source code will contain appropriate useful comments that explain what the code does. The source code will be laid out with only minor inconsistencies.  **Structure**: Where appropriate, the code will show some thought into the development of structural units (like classes) resulting in clear code, but perhaps not really facilitating code reuse. Associated Graphical User Interfaces and websites will show thought in their structuring, with appropriate separation of functionality (HTML; CSS; JavaScript).  **Validity**: The code is expected to contain some checks to prevent errors as a result of users not using the program correctly.  **Efficiency**: The code may be inefficient but it should still run in a reasonable time frame.  **Functionality**: The code will include the core functionality, with only minor issues. Associated Graphical User Interfaces and websites will work and provide access to the core functionality. |
| **Pass** | **Readability**: The code/software may have limited documentation. There may be little evidence that thought has gone into code structuring or standards. There may only be minimal inline comments. There may be an attempt to lay out the code neatly, but there may be some inconsistency. Any asked for ancillary documentation may lack detail or contain errors.  **Structure**: The code may only follow broad structures associated with the coding languages or suggested as starting points for the assessment. Where there is an expectation of code separation, much of the code may nevertheless be in a single file or block. Graphical User Interfaces and websites may not function well or there may not be a clear separation of functionality (HTML; CSS; JavaScript).  **Validity**: Minor errors may be encountered at runtime.  **Efficiency**: The code may be inefficient and run times may be unnecessarily long.  **Functionality**: There will be evidence of an attempt to include the core functionality, although minor issues may have been encountered. Key elements of Graphical User Interfaces and websites may work, but these may have multiple issues. |
| **Fail** | **Readability**: The code/software will have little or no documentation. Inline comments will be minimal or non-existent, or may contain errors or misunderstandings or be generally not very informative. The code layout may be somewhat confusing and difficult to read. Any ancillary documentation may be of very limited use. Issues encountered in developing the code/software may not be explained well if at all.  **Structure**: The code is likely to be unstructured and may, for example, be in a single block. For Graphical User Interfaces and websites, key links between different sets of code or functionalities may be broken.  **Validity**: The code may fail to run due to poor implementation.  **Efficiency**: The code may be very inefficient.  **Functionality**: There may be evidence of an attempt to include the core functionality, but this will not work. Graphical User Interfaces and websites may be confusing, distracting, and not adhere to standards. |

# Feedback Sheet

In general, your overall mark will reflect the average of the categories below, however please note that the grades for the individual components are nevertheless indicative only and the overall mark may vary from this. For example, a lecturer may decide that a very nice but minor piece of code warrants a higher mark in total, despite not making a huge difference to any one of the individual categories. Please read the detailed feedback in order to understand your mark.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | High Distinction | Distinction | Merit | Pass | Fail |
| Readability |  |  |  |  |  |
| Structure |  |  |  |  |  |
| Validity |  |  |  |  |  |
| Efficiency |  |  |  |  |  |
| Functionality |  |  |  |  |  |
| Overall |  |  |  |  |  |

Feedback

Areas to especially prioritise

|  |  |
| --- | --- |
| Marker | Provisional mark |
|  |  |

*Any mark given here is provisional and subject to moderation by the School’s Board of Examiners. This is to ensure comparable marking standards for all students. In a minority of individual cases moderation can lead to either the raising or lowering of the provisional marks.*