Computer Code Marking Criteria

Your program will be assessed against the following criteria:

* **Readability**: *The ease with which your code can be understood by a human. This includes the abundance and clarity of comments, naming conventions, and the use of code indentation, whitespace, etc. to ease understanding.*
* **Structure**: *The degree to which the code has been organised into relevant blocks, files and other structures as appropriate.*
* **Validity:** *The absence of logical or functional errors and the presence of appropriate tests and internal checks.*
* **Efficiency**: *The extent to which the code reduces unnecessary computation and limits memory usage.*
* **Functionality**: *The overall ability of the software to solve a specified problem or help the user achieve their ends, including the usability and aesthetics of Graphical User Interfaces and websites.*

“Ancillary documentation” in the table below is any essay about the development process, diagrams, etc. that you are asked for in addition to the code. You will not always have been asked for this. Where you have not been made aware of terms (for example, “loose coupling”), either in lectures or further reading, you will not be expected to match these standards, though you are, of course, welcome to. What the criteria mean will also change as you learn, so, for example, early on “good structure” may mean code that is clearly organised within a single block, while later it may mean placing code into procedures or classes, once you know those terms. It should be noted that the code not compiling or running at the final stage does not *necessarily* doom the code to a low mark, provided the bulk of the code would work were issues resolved.  The final mark may result from a combination of elements from across the marking scheme.

|  |  |
| --- | --- |
| **Grade (UG / TPG)** | **Typical Criteria** |
| **Super First / High Distinction** | **Readability**: The code will have the best of professional standards in documentation; a high level of useful comments; be perfectly laid out; and where appropriate ancillary documentation will detail a professional [development process](https://en.wikipedia.org/wiki/Software_development_process) and suitable [structural diagrams](https://en.wikipedia.org/wiki/Unified_Modeling_Language). The program itself will come with structured help files embedded throughout the system where necessary (if appropriate).  **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 will 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 influenced of the [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; JS) based on best practice.  **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.  **Efficiency**: The code will be close to optimal efficiency in terms of both memory and processor usage.  **Functionality**: The code will show the core functionality, enhanced with additional functionality where appropriate, with a concentration on ensuring that users cannot make mistakes and/or that mistakes are resolved with minimum input. There will, where appropriate, be a well-designed set of [usability](https://en.wikipedia.org/wiki/Usability) tests detailed in any ancillary documentation. Associated Graphical User Interfaces and websites will have high aesthetic and usability standards, potentially showing the influence of [Human-Computer Interaction](https://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction) literature. Overall, the code will be of a professional release standard, or close. |
| **First / Distinction** | **Readability**: The code will meet professional standards in documentation; have a range useful comments; be almost perfectly laid out; and where appropriate ancillary documentation will detail a professional [development process](https://en.wikipedia.org/wiki/Software_development_process) and suitable [structural diagrams](https://en.wikipedia.org/wiki/Unified_Modeling_Language). The program itself will come with at least one help file where necessary.  **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, in line with best-practice examples. For websites this will include good structuring of associated directories and files, with a solid separation of functionality (HTML; CSS; JS).  **Validity**: The code will show some evidence of additional checking, especially to prevent errors arising during the running of the program.  **Efficiency**: The code will show that some thought has been put into efficiency and these are largely successful.  **Functionality**: The code will show the core functionality, enhanced with additional functionality where appropriate, with a concentration on ensuring users cannot make mistakes and/or that mistakes are resolved with minimum input. There will, where appropriate, be some discussion of [usability](https://en.wikipedia.org/wiki/Usability) in any ancillary documentation, and usability will be a core consideration in any associated Graphical User Interfaces and websites. Both GUIs and websites will display clear thought around the development of aesthetics. Overall, the code will be of a near professional release standard with only minor usability issues. |
| **IIi / Merit** | **Readability**: The code will have appropriate documentation, but not full level; it will have inline comments; be laid out with only minor errors; and where appropriate ancillary documentation will detail the thought processes behind the code and include [structural diagrams](https://en.wikipedia.org/wiki/Unified_Modeling_Language) with only minor non-standard elements.  **Structure**: Where appropriate, the code will show some thought into the development of structural units (like classes), but with a concentration on clarity not code reuse. Associated Graphical User Interfaces and websites will show thought in their structuring, with appropriate separation of functionality (HTML; CSS; JS).  **Validity**: The code will include standard elements of checking and elements to prevent errors.  **Efficiency**: The code will not be inefficient.  **Functionality**: The code will show the core functionality, with only minor issues. Associated Graphical User Interfaces and websites will work to allow core functionality, rather than getting in its way. |
| **IIii / Pass** | **Readability**: The code will have limited documentation, potentially without proper thought into its structuring to standards. It will have minimal inline comments; and will show an attempt at laying out the code to standards. Any asked for ancillary documentation will be of limited detail and any structural diagrams will largely fail to follow standards.  **Structure**: The code will 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, most of the code will nevertheless be in a single file or block. For Graphical User Interfaces and websites, when separation of functionality is regarded as good practice, this will be limited.  **Validity**: The code will include minor errors at runtime or compilation.  **Efficiency**: The code will be inefficient.  **Functionality**: The code will show an attempt at the core functionality, with only minor issues. Key elements of Graphical User Interfaces and websites will work against core functionality, and aesthetics will be distracting and/or disjointed and/or break well-known standards to no advantage. |
| **III / Fail for Masters** | **Readability**: The code will have no documentation. It will have very minimal inline comments that may display some misunderstanding; layout will be chaotic and difficult to read. Any ancillary documentation will be of very limited detail and structural diagrams may be missing.  **Structure**: The code will not be structured and may, for example, be in a single block where this is inappropriate for the language in question. For Graphical User Interfaces and websites, key links between different sets of code or functionalities may be broken.  **Validity**: The code will include significant errors at runtime and/or compile time and/or work to an incorrect algorithm.  **Efficiency**: The code will be very inefficient.  **Functionality**: The code will make an attempt at some elements of the core functionality, but will largely fail to get it working. Graphical User Interfaces and websites will lead the users to expect functionality they fail to deliver, and/or be otherwise confusing. GUI and website aesthetics will be distracting and/or disjointed and give little or no thought to well-known standards. |
| **Fail for Undergraduate and Masters** | **Readability**: The code will have no documentation or commenting, or comments may show significant misunderstanding suggesting the code has been derived from lectures without understanding. Layout will be chaotic and unpleasant to read. Any ancillary documentation will be missing  **Structure**: The code will follow not structuring and may, for example, be in a single block where this is inappropriate for the language in question. For Graphical User Interfaces and websites, key links between different sets of code or functionalities may be broken.  **Validity**: The code will include significant errors at compile time and/or with an incorrect algorithm.  **Efficiency**: The code will be very inefficient.  **Functionality**: The code will make little attempt at any elements of the core functionality, and/or will do a completely different job, and/or be a random collection of code fragments. Graphical User Interfaces and websites will be broken and confusing. |

Computer Code Mark and Feedback Sheet

For information about the criteria code is marked against, see the associated *Computer Code Marking Criteria*.

General Class Comments

Individual Feedback

*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.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Undergrad → | Super first | First | IIi | IIii | III | Fail |
| Masters     → | (High) Dist | Distinction | Merit | Pass | Fail | 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.*