Assignment One – Detailed Code Style Criteria

Criteria	**Standard**		
Readability	**Proficient**	**Competent**	**Novice**
• Program Structure	Modular blocks are used to highlight code logic, making it easy to understand.	Some modular blocks are used to highlight code logic, along with some less clear code structure.	Poor code structure is used, which makes the code difficult to read.
• Identifier Names	All identifier names are informative, and well chosen, increasing readability of the code.	Most identifier names are informative, aiding code readability to some extent.	Many identifier names are not informative, detracting from code readability.
• Named Symbolic Constants	All, non-trivial, fixed values (literal constants) in the code are represented by informative, named (symbolic) constants	Most, non-trivial, fixed values (literal constants) in the code are represented by informative, named (symbolic) constants.	Only some, non-trivial, fixed values (literal constants) in the code are represented by informative, named (symbolic) constants.
Algorithmic Logic			
• Single Instance of Logic	Almost no code has been duplicated in your program. You have well designed functions with appropriate parameters to modularise your code.	Some code has been duplicated in your program. You have used some functions to modularise your code.	Large amounts of code are duplicated in your program. You have made poor use of functions to modularise your code.
• Variable Scope	Variables are declared locally in the functions in which they are needed. Global variables have not been used.	At most one global variable has been used. Or, there are a few unnecessary local variables in functions.	Global variables have been used, reducing the clarify of function logic.
• Control Structures	Logic is structured simply and clearly through good use of control structures.	A small number of control structures are unnecessarily complex.	Many control structures are poorly designed (e.g. excessive nesting, overly complex conditional logic, loops with multiple unnecessary exit points, ...).
Documentation			
• Comment Clarity	Almost all comments enhance the compre-hensibility of the code. Comments almost never repeat information already apparent in the code.	A few comments are unnecessary to the comprehension of the code. Alternatively, a few comments are overly verbose reducing the ease with which code can be comprehended.	Many comments are unnecessary to the com-prehension of the code. Alternatively, many comments are overly verbose reducing the ease with which code can be comprehended.
• Informative Docstrings	All docstrings are accurate and informative, and clearly show how parameters and return types should be used.	Almost all docstrings are accurate and reason-ably clear descriptions of how the module or function is to be used. Almost all parameters and return types are described clearly.	Several docstrings are inaccurate or unclear, or absent. Some parameters and return types are unclear.
• Description of Logic	All important or complex blocks of logic (e.g. major loops or conditionals) are almost always clearly explained or summarised. Almost no stating of the obvious (e.g. a loop iterates over a sequence).	Most important or complex blocks of logic (e.g. major loops or conditionals) are usually clearly explained or summarised. Almost no stating of the obvious (e.g. loop iterates over a sequence).	Some important or complex blocks of logic are explained or summarised poorly (e.g. descrip-tion is unclear or restates code). Alternatively, some unimportant blocks of code are given excessive coverage in the comments.