Code Review Rubric

Code Quality Elements	Needs Improvement	Approaching Mastery	Mastery
Readability and Formatting Variable naming and casing Line length and complexity Formatting and indentation Explanations in comments	❑ Unclear/arbitrary variable names ❑ Style is sometimes inconsistent ❑ Lines are always too complex ❑ Inconsistent formatting/indentation ❑ Few or no comments to explain complex or confusing code	❑ Descriptive variable names ❑ Style is always consistent ❑ Lines are often too complex ❑ Readable formatting/indentation ❑ Several comments to explain complex or confusing code	❑ Clear, semantic variable names ❑ Style always follows conventions ❑ Lines only contain a single idea ❑ Consistent formatting/indentation ❑ Complex code is always explained with comments when appropriate
Organization and Modularity Modularity and coupling Use of abstraction Side effects of functions	❑ Code contains large monolithic or tightly coupled functions and/or classes that could be separated ❑ Limited or no use of abstraction ❑ Functions use global variables	❑ Code is separated into functions and/or classes but may be tightly coupled causing ripple of changes ❑ Some use of abstraction ❑ Few functions cause side effects	❑ Code is separated into functions and/or classes with different, clear responsibilities and loose coupling ❑ Abstraction used whenever helpful ❑ All functions avoid side effects
Built-ins & Standard Library / Language Conventions Uses existing functions/classes Follows language conventions	❑ Several built-ins and standard library functionality are reinvented without any customization or justification ❑ Violates language conventions	❑ Occasional use of built-ins and standard library shows need for more exposure and/or research ❑ Few cases of reinvention could be simplified using built-ins and standard library standard	❑ Built-ins and standard library are used whenever possible ❑ Follows language conventions and idioms
Effectiveness of Solution Does it solve the problem?	❑ Solves some typical input cases ❑ Does not solve any edge cases	❑ Solves most typical input cases ❑ Solves some obvious edge cases	❑ Solves all typical input cases ❑ Solves all known edge cases
Testing and Error Handling Testing solution robustness Handling errors/exceptions	❑ Minimal or no automated testing ❑ Test inputs are simplistic or naive ❑ Minimal or no exception handling	❑ Tests cover typical input cases ❑ Test inputs are varied and creative ❑ Handles some errors/exceptions	❑ Tests cover all typical input cases ❑ Tests cover all known edge cases ❑ Handles several errors/exceptions
Algorithmic Complexity Efficient use of resources Scalability with large inputs An explicit statement of time and space complexity	❑ Code often repeats redundant operations or uses brute force ❑ High algorithmic complexity that does not scale with large inputs ❑ Does not state time and space complexity	❑ Some code repeats redundant work, but with minimal impact ❑ Low algorithmic complexity that avoids brute force approaches ❑ Incorrectly / incompletely state time and space complexity	❑ Repeated work is often avoided to save time and memory resources ❑ Optimal algorithmic complexity that scales well with large inputs ❑ Correctly states time and space complexity

Inspired by MakeSchool's Code Rubric