Rubric for Computer Programming

Christopher League, November 2013

Can be adapted for assignments to produce software artifacts in CS102, 117, 120, 130, 155, 156, 161, 164, 601, 631, 653, 673, 690, and others. As stated on AAC&U VALUE rubrics, “evaluators are encouraged to assign a **zero** to any work…that does not meet beginning (cell one) level performance.”

	Advanced **4**	**Proficient** **3**	**Approaching Proficiency** **2**	**Beginning** **1**
Syntax Ability to understand and follow the rules of the programming language.	Program compiles and contains no evidence of misunderstanding or misinterpreting the syntax of the language.	Program compiles and is free from major syntactic misunderstandings, but may contain non-standard usage or superfluous elements.	Program compiles, but contains errors that signal misunderstanding of syntax – such as the semi-colon in `if(exp);{}`	Program does not compile or (in a dynamic language) contains typographical errors leading to undefined names.
Logic Ability to specify conditions, control flow, and data structures that are appropriate for the problem domain.	Program logic is correct, with no known boundary errors, and no redundant or contradictory conditions.	Program logic is mostly correct, but may contain an occasional boundary error or redundant or contradictory condition.	Program logic is on the right track with no infinite loops, but shows no recognition of boundary conditions (such as **<** vs. **<=**)	Program contains some conditions that specify the opposite of what is required (less than vs. greater than), confuse Boolean AND/OR operators, or lead to infinite loops.
Correctness Ability to code formulae and algorithms that reliably produce correct answers or appropriate results.	Program produces correct answers or appropriate results for all inputs tested.	Program produces correct answers or appropriate results for most inputs.	Program approaches correct answers or appropriate results for most inputs, but can contain miscalculations in some cases.	Program does not produce correct answers or appropriate results for most inputs.

	Advanced **4**	**Proficient** **3**	**Approaching Proficiency** **2**	**Beginning** **1**
Completeness Ability to apply rigorous case analysis to the problem domain.	Program shows evidence of excellent case analysis, and all possible cases are handled appropriately.	Program shows evidence of case analysis that is mostly complete, but may have missed minor or unusual cases.	Program shows some evidence of case analysis, but may be missing significant cases or mistaken in how to handle some cases.	Program shows little recognition of how different cases must be handled differently.
Clarity Ability to format and document code for human consumption.	Program contains appropriate documentation for all major functions, variables, or non-trivial algorithms. Formatting, indentation, and other white space aids readability.	Program contains some documentation on major functions, variables, or non-trivial algorithms. Indentation and other formatting is appropriate.	Program contains some documentation (at least the student's name and program's purpose), but has occasionally misleading indentation.	Program contains no documentation, or grossly misleading indentation.
Modularity Ability to decompose a problem into coherent and reusable functions, files, classes, or objects (as appropriate for the programming language and platform).	Program is decomposed into coherent and reusable units, and unnecessary repetition has been eliminated.	Program is decomposed into coherent units, but may still contain some unnecessary repetition.	Program is decomposed into units of appropriate size, but they lack coherence or reusability. Program contains unnecessary repetition.	Program is one big function or is decomposed in ways that make little sense.