Using Mastery Learning Ideas to Assess Student Work

Assessing the skills of my students is the hardest part of teaching coding and/or programming. Assessing a student's work could boil down to the question, "did the program output the right answer?" However, I believe students should learn that good programming involves more elements than just producing the "right" answer. What are these objectives and how do I address them? The following questions allow me to efficiently evaluate students' programs:

1. Is the code clear? (Does it contain enough comments?)
2. Does it run without errors?
3. Is the output correct?
4. Does it deal with "junk" input and questionable output well enough?

While each question addresses one of my learning objectives, many students only consider question 3 when programming. I use the ideas behind mastery learning to encourage students to consider all the elements of a good program. After students submit their programs, I give them feedback and a score based on the criterion above and allow them to resubmit as many times as needed to meet all the objectives (with the stipulation they make progress between resubmissions). This encourages student engagement, allows students to take risks and learn more, and gives me an opportunity to analyze the finer details of the students' coding practices.

Allowing a back and forth between the students and me naturally engages the student. Undergraduate students today place great importance on grades; therefore, when given an opportunity to improve their grade, they usually take it. During the semester, almost every student stops by my office to discuss their assignments and seem genuinely engaged with understanding the programming methods involved. I have even seen debates erupt in class among students about the best way to code solutions to assignments.

Allowing students to resubmit work with no penalty encourages students to take time doing the assignment right and not just "good enough". With time to go down blind alleys and try new things, students naturally explore more of the programming languages and learn tricks that otherwise would go undiscovered. People learn by doing and allowing students to modify and improve their work gives them the freedom to explore all the wrong and some unexpected ways to program.

Giving students multiple chances for feedback allows me to assess their work beyond the obvious questions of "does it run?" and "are the results correct?" I give myself the room to ensure that students create readable code and test program input/output. These two tasks do not come naturally for students and need to be practiced and reinforced by instruction. If students remain in technical fields beyond graduation, they will invariably pass their code onto

a colleague. Passing off readable code with clear indentation, comments, and input testing will set them apart from others in their workspace (speaking from experience). Giving students ample opportunity to practice increases the chances of them carrying this habit forward and making an impression in their chosen field.

Assessing students programming skills can be hard and time consuming, but I have found that accepting resubmitted work engages students, allows them to take risks, and allows me to assess the finer details of their programs.