Coderunner: A Tool for Assessing Computer Programming Skills

By Richard Lobb and Jenny Harlow,
The University of Canterbury

How should we assess programming skills? Asking students to write code in a traditional hand-written exam can produce results like those in Figure 1. It is nearly impossible to meaningfully grade such code. With sufficient effort one can get some idea of whether the general idea is correct, but to assess programming skill we need much more than this. For example, there will almost certainly be errors in the code; how do we know whether the student would be able to correct those errors or not?

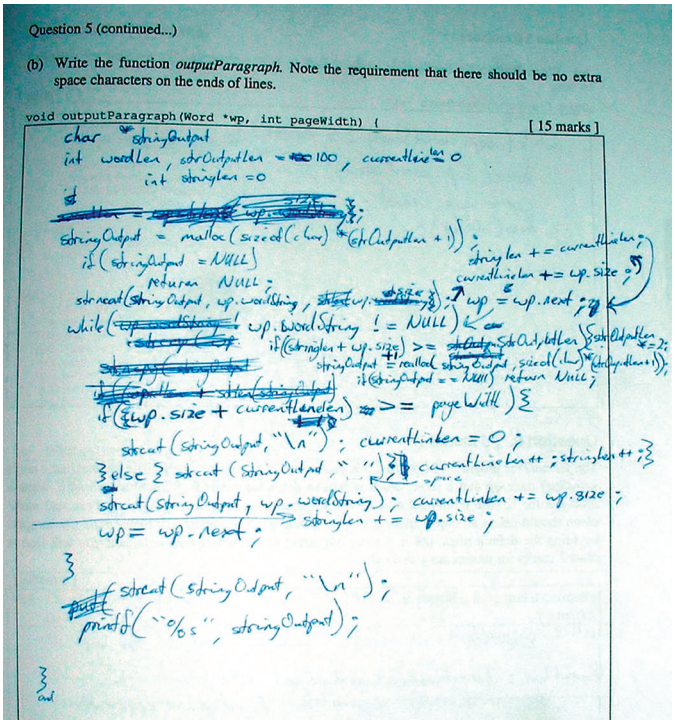
We pity the marker faced with grading code like that in Figure 1, but shouldn't we be even more sympathetic to the student who had to write it? Modern programming is an on-line process involving interaction between the programmer and the computer. Few programmers get their code correct on the first try—testing and debugging is an integral part of the programming process. To assess programming skills, we should provide students with an authentic programming environment in which they can develop and test their code. Only then is it fair for us to run their code and use correctness tests as our measure of their ability.

Examinations are just one aspect of assessment. In introductory programming courses we usually also assess laboratory and assignment work.

This article introduces a new tool that helps with all these different assessment requirements.

INTRODUCING CODERUNNER

CodeRunner [1,2,6] is a free and open-source Moodle [3] question-type plug-in that lets teachers set questions where the answer is program code. Students develop and test their code using a normal development environment and submit the code to CodeRunner through a web browser only when they believe it is correct. A key assumption behind CodeRunner is that the quiz in which the questions appear is running in Moodle's *adaptive* mode, which gives students immediate feedback on the correctness of their answer and allows them to resubmit for a penalty.


Question 5 (continued...)

(b) Write the function *outputParagraph*. Note the requirement that there should be no extra space characters on the ends of lines.

```
void outputParagraph(Word *wp, int pageWidth) {
```

[15 marks]

Figure 1: What grade is this worth?

CodeRunner provides a flexible penalty regime that allows authors to tailor the grading to the context; for example, one might have some questions that apply no resubmission penalties, some that allow one or two free submissions followed by an increasing 10% penalty for each subsequent wrong submission, and others that apply a 100% penalty for even one wrong submission.

CodeRunner can support any text-based programming language. Built-in question types are available for C, Java, Python, PHP, JavaScript and Octave (Matlab). CodeRunner is highly scalable: questions can range from simple fill-in-the-blank coding questions through to fairly major assignments. Because it is just another Moodle question type, test or exam quizzes can mix CodeRunner questions with other computer-graded question types like multi-choice, numeric, short answer, or matching. One can even incorporate essay questions graded by humans. In a learning context such as laboratories the "quizzes" may include "description" questions, which are actually just tutorial material, for example, an introduction to if-statements or loops. The quizzes can then become the primary learning medium. Being web-based, they can be done by students from home or after-hours, freeing the course from many of the constraints imposed by scheduled laboratories, which can instead become help sessions for students who need the extra support.

CodeRunner is used at the University of Canterbury for teaching programming courses in Python, C, Octave and Matlab. CodeRunner is particularly well suited to introductory programming courses, for which students need lots of practice with small programming problems that teach the different language constructs and techniques. However, CodeRunner has also been used at higher levels, for example in theoretical computer science papers for testing things like finite-state automata and compiler construction, in an artificial intelligence course for Clojure programming, and in a web programming course for assessing student-authored websites.

CODERUNNER IN ACTION

Figures 2 and 3 show a simple Python CodeRunner question that asks the students to write a function *squares(n)* that returns a list of the squares of all integers from 1 to n inclusive. Figure 2 shows what the student sees after an incorrect submission. Figure 3 shows the state after a correct submission. Here are some key things to note.

- The student gains immediate feedback by clicking the *Check* button.
- The feedback is a table that shows the tests that were used, the expected output from each test of the student's function, and the actual output. Green ticks denote correct outputs, red crosses wrong ones.
- The feedback panel is colored red if any of the outputs are wrong (the default "all or nothing" grading method) and zero marks are awarded. When the student resubmits a correct answer (Figure 3) the entire panel goes green and a mark of 100% is awarded (less whatever penalty has accumulated, in this case 10%).
- In this example, there are hidden test cases to prevent students synthesizing an answer that works just for the supplied tests.

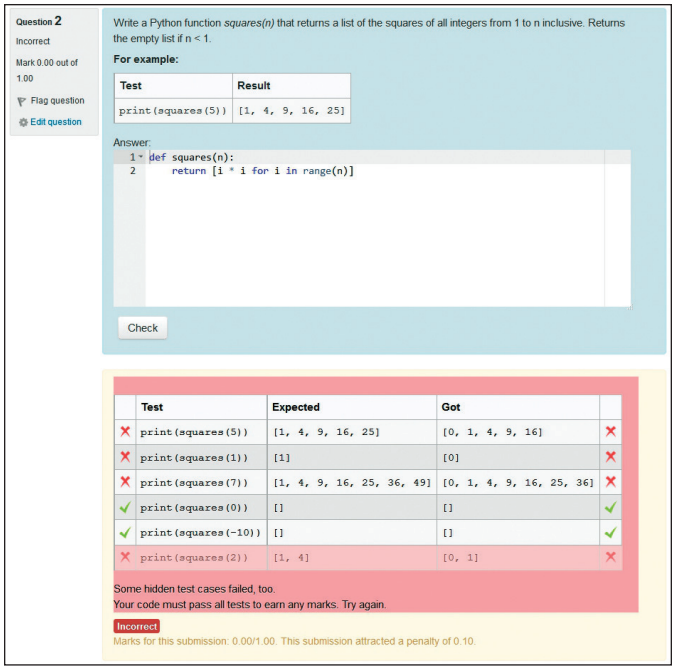


Figure 2: The simple Python question, wrongly answered.

The format of the feedback was inspired by Nick Parlante's *codingbat* website [4]. The green ticks and green feedback panel coloring seems to be extraordinarily motivating. Students work very hard to get their answers right the first time and obvious signs of delight often accompany the appearance of a green result table in the labs. Even question developers get satisfaction from correctly answering their own trivial problems!

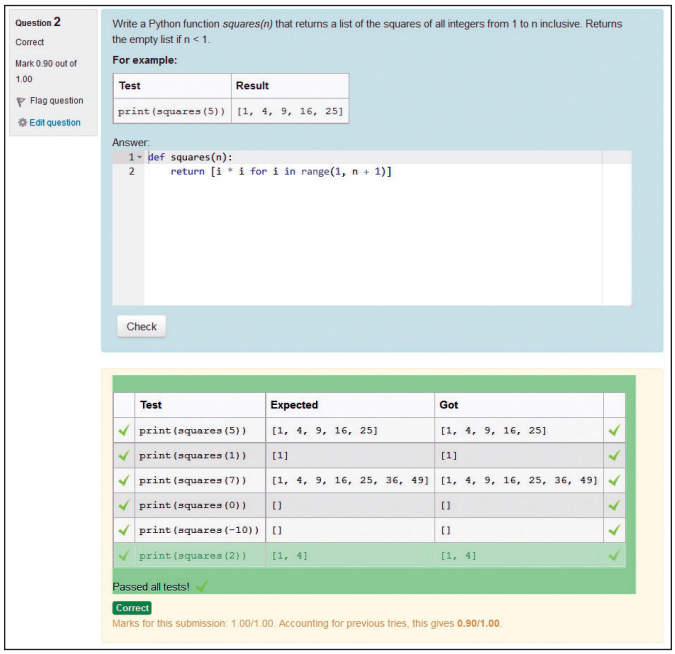


Figure 3: The simple Python question, correctly answered.

Students work very hard to get the green ticks and almost never move on having been marked wrong on a question—they take the time to get it right before continuing.

WHAT'S DIFFERENT ABOUT CODERUNNER?

Computerized testing environments are nothing new but, we suggest, the Moodle/CodeRunner combination is fundamentally different from most such environments.

- CodeRunner is a free open-source plug-in for the free open source Moodle learning platform [3], which is used worldwide by thousands of universities and schools.
- The Moodle/CodeRunner combination can be used throughout the course in laboratories, tests, assignments, and even final exams. Assessment, grade recording, and distribution of course material are thus all consolidated on the same platform.
- Because CodeRunner is integrated with the Moodle learning platform, CodeRunner questions can be intermingled with tutorial information ("description" questions) and other question types.
- CodeRunner supports an essentially unlimited range of programming languages and is very flexible in terms of the type of question that can be asked (see below), the penalty structure used, and even the form of feedback displayed.
- An added benefit of the consolidated approach described above is that students become very familiar with the environment so that the stress of online tests and examinations is considerably reduced.

STAFF AND STUDENT FEEDBACK ABOUT CODERUNNER

Students have generally been very positive about CodeRunner. They particularly value the immediate feedback. In a laboratory context, the instant grading of each question helps maintain motivation. Students work very hard to get the green ticks and almost never move on having been marked wrong on a question—they take the time to get it right before continuing. And students report to us that they appreciate being able to track their level of understanding throughout the laboratory.

In an exam context, students also report finding the immediate feedback helpful as it removes the uncertainty of not knowing how they're going and they appreciate leaving the exam room knowing their grade. As one student said "Instant quiz servers are great! Immediate test marks are awesome!"

Teachers have also been very positive about CodeRunner. Unsurprisingly, they appreciate having to do little or no test and exam marking, particularly with large classes. More surprisingly, teachers appear to actually enjoy designing CodeRunner questions. Devising and implementing a good question to test some aspect of programming is both challenging and rewarding. Developing a new *question type* for a whole class of programming problems generally involves some coding by the question author and proves particularly enjoyable. Contrast that with writing questions for written examinations, which is rarely if ever thought to be fun.

WHAT SORT OF QUESTIONS CAN BE ASKED?

Every CodeRunner question is an instantiation of a *prototype* question, which essentially defines the "type" of the question. A set of built-in prototypes define a range of "write a function," "write a program," or "write a class" questions for the standard languages, but question developers can define their own prototypes to provide extra functionality.

The prototype defines, by means of a template, what program should be executed for a given student answer and test case. The resulting program is compiled and run, and the output from that run is compared with the expected output to determine if the code passes that particular test. For security, execution of the wrapped student code usually takes place on a separate machine, called the *sandbox server*.

The use of a customizable template to define the program to be executed provides great flexibility. It allows the question author to use the student's answer in many different ways. A major application of this flexibility is some form of pre-processing that validates the student submission before running it. This can be illustrated in two important ways.

- *Style-checking students' submissions.* For Python we can enforce compliance with the industry-standard *pylint* style checker [6] before the code is accepted for execution. For Octave (Matlab) question types we have written templates that apply local style rules and can enforce limits on both program and function size.
- *Enforcing and/or restricting the use of particular programming constructs.* For example, we have questions in several languages of the form "rewrite the following program using a *while* loop instead of a *for* loop." The student's code is rejected without being run if the pre-processer detects any *for* loops present.

The language used for the prototype can be different from that used to execute the student's submission. For example, one of our local Octave style checkers uses a template in Python to check students' code before submitting it to Octave for execution. This distinction between the template language and what is being checked is used in most of the following examples, all of which exploit the flexibility of templates.

- A question in a compiler-construction course where the student's submission is a simple compiler that outputs Java Virtual Machine (JVM) code. The template code first executes the student's code on a test program and then runs the output from that on the JVM.

- A question in which the student's answer is just a URL referencing a web page they have built on a separate server. The template code then performs tests on the referenced web page.
- A question type in which a student submits a textual description of a Finite State Machine (FSM) and the question developer's code (written in Python) validates and grades the FSM's behavior.
- A *python-tkinter* question type in which the student's answer involves a graphical user interface (GUI). The template code for this question includes a mock implementation of the small *tkinter* subset taught in the course, allowing the question author to test the behavior of the GUI in various ways.

Normally, the correctness of a student's submission is validated by comparing the actual output from the run with the expected output for each test, but it is also possible to incorporate the grading process into the template itself, as is done in the FSM example above.

In principle any question that can be graded by a computer can be posed as a CodeRunner question, though CodeRunner is best suited to dealing with exercises where the tests can be presented to the students in tabular form.

CODERUNNER IN THE INTRODUCTORY PROGRAMMING COURSE

CodeRunner was initially developed for use in our introductory programming course COSC121, taught in Python, though it has since spread into several other courses in computer science, engineering and mathematics. While originally introduced purely as a way of assessing laboratory work, CodeRunner quizzes have now become the primary way in which the COSC121 course is taught and assessed. A departmental Moodle server has been set up just to run the quizzes.

Currently, there are a number of assessments within COSC121.

- *Ten lab quizzes*, one per week, each worth 1% of the course grade. Having these on the web has had some unexpected spin-offs. Students are much more able to work at home, easing the pressure on scheduled laboratories, which become more like help sessions for students. Maintenance and polishing of these "laboratories" is much easier, too. If an error is detected during a scheduled lab, we simply correct it there and then and all students immediately see the updated version.
- *Four "assignment superquiz" quizzes*, which replace the traditional single assignment we used to set. Together these contribute to 20% of the course assessment. Having a staged sequence of tasks is less daunting for weaker students than a single large assignment and has resulted in a higher participation rate in the final exam. Assignment quizzes and lab quizzes all require that students submit code that is accepted by the Python style checker, *pylint* [5]. However, since no program can evaluate how well the variables and functions are named, or how appropriately chosen the functions are, we also still have humans evaluate the quality of the final superquiz submissions, imposing a small style penalty for poor code.
- *Occasional optional drill quizzes*, which contribute nothing to the grade of the course but provide students with extra practice.
- *A mid-semester test*. This test, worth 15% of the course grade, is an invigilated 1.5-hour test, mostly using questions from the laboratories and the drill quizzes. Questions are randomized, each selected from a small pool of options, so that each student gets a different test but with the spread of questions and overall difficulty being very similar for all.
- *A final exam, worth 55% of the grade*. This is an invigilated 3-hour exam, set in the laboratories. Both the exam and the test use a locally-developed environment that provides the same program development tools as in the laboratories but which prevents web access or other outgoing connections, except to the quiz server.

While originally introduced purely as a way of assessing laboratory work, CodeRunner quizzes have now become the primary way in which the COSC121 course is taught and assessed.

Having an online exam rather than a hand-written one can yield some interesting insights. Moodle provides statistics on how students perform on each question, so we can immediately spot topic areas that might warrant further teaching effort in the future and also questions that prove problematic (e.g., that appear to discriminate against good students). The progress of individual students through an exam—we call this a *mark or grade trajectory*—can be plotted, as in Figure 4 below. This shows that most students were close to their final mark just one hour into the three hour exam, but there were also a small number of slow steady students whose mark increased steadily throughout the exam.

LIMITATIONS OF CODERUNNER

Although CodeRunner has proved very valuable over a wide range of assessment activities, there are some fundamental limitations.

- Code-quality tools like *pylint* have proved very valuable in raising style awareness and improving code quality but abuses of the style rules still occur, and the quality of comments and identifiers cannot be assessed by computer. Hence COSC121 still reserves some small portion of its in-course assessment for human-graded code quality.

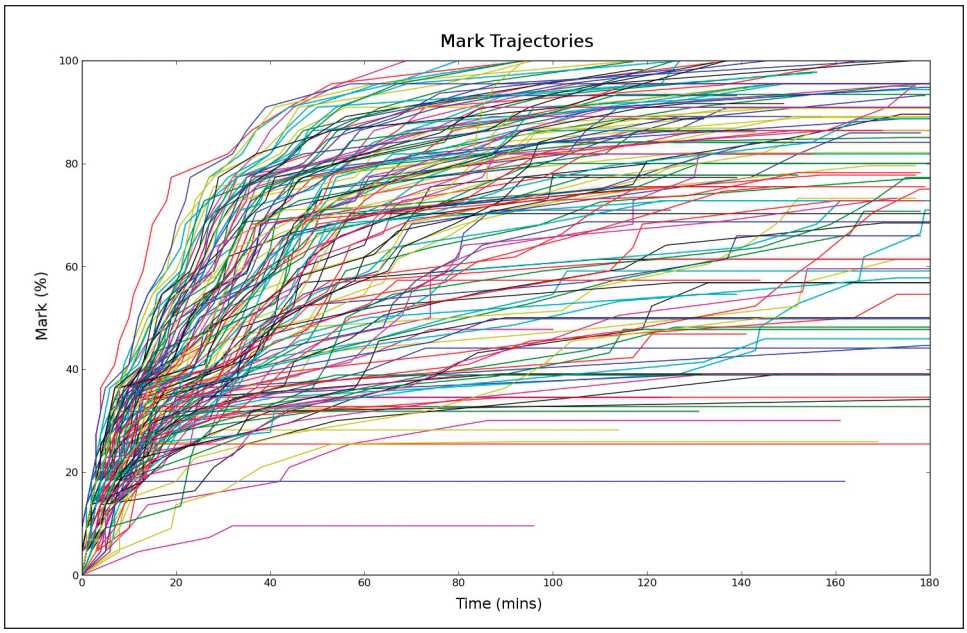


Figure 4: Grade trajectories for all students in a CodeRunner-based exam.

- CodeRunner is best suited to relatively simple tasks with a precise specification. Although any question in which the answer can be assessed by a computer program can in principle be cast as a CodeRunner question, the time required to write a grader for a complex task often makes the approach not worthwhile, particularly for smaller classes.
- The answer to a CodeRunner question must be a single block of text (in most of the examples in this article the "text" is code). This limits CodeRunner's applicability in more advanced programming courses where multiple files are involved
- Although the author of a CodeRunner question can in principle generate any form of feedback, even including graphics, the normal results display is tabular, with one table row per test case and with simple exact matching of expected output with actual output. If test cases require large blocks of code or if there is considerable output from each test, the sheer size of the results display can make it difficult for students to see why their answer is being marked wrong.
- Assessing questions with graphical output is problematic, though we have started to make some steps in that direction. We have built a mock of the Python GUI toolkit *tkinter* for grading GUIs in COSC121 and have questions that assess the correctness of graphs generated by calls to Matlab's graphing library. But assessing the correctness of an image or even the output from a turtle graphics program is hard or even impossible.
- Finally it must be mentioned that writing good quiz questions and good tests can be very time consuming even for low-level programming courses. Sharing question databases with other teachers would be a huge help here and also in setting up some sort of repository to facilitate such sharing is a future priority.

CONCLUSION

Moodle/CodeRunner quizzes have transformed several of our programming courses. The ability to mix traditional style questions with coding questions has proved very valuable for both laboratory and examinations. Staff and students have both been very positive in their response to CodeRunner. Staff enjoy using CodeRunner and are pleased to be able to directly assess the actual skill they're trying to teach (programming). They are particularly pleased not to have to grade hand-written code. Students are very positive about the instant feedback they get. They like the intermingling of tutorial material, normal Moodle questions, and CodeRunner questions in laboratories and they respond well to being able to take tests and exams with most of their standard tools on hand. ❖

References

1. CodeRunner demo site; http://coderunner.org.nz. Accessed 2015 May 21.
2. CodeRunner repository; https://github.com/trampgeek/CodeRunner. Accessed 2015 May 21.
3. Moodle; http://www.moodle.org. Accessed 2015 May 21.
4. Nick Parlante's CodingBat site; http://codingbat.com. Accessed 2015 May 21.
5. PyLint Python style checker; http://www.pylint.org/. Accessed 2015 May 21.
6. Video on CodeRunner; https://www.youtube.com/watch?v=I6AO5CobNyo. Accessed 2015 May 21.

Richard Lobb
Department of Computer Science and Software Engineering,
The University of Canterbury

Jenny Harlow
School of Mathematics and Statistics
The University of Canterbury

DOI: 10.1145/2810041