Developing a Skills-Based Framework for Assessments in Programming Courses

Elizabeth Pankratz

School of Philosophy, Psychology, and Language Sciences, University of Edinburgh
elizabeth.pankratz@ed.ac.uk

In This Chapter

To learn to program is to develop a new set of skills. In this chapter, I'll discuss an approach to course design and assessment that brings skill development to the fore: the skills-based assessment framework. I'll explore what makes this framework particularly good for teaching programming to new programmers, especially ones facing cultural, structural, and institutional disadvantages. And I'll share a case study from my own context: I work as a Lecturer in Psychology at the University of Edinburgh, teaching undergraduate Psychology students how to do statistics using R. With my colleagues, I have developed an introductory course for these students using the skills-based framework, and our experience will be the focus of the final part of this chapter.

What Is Skills-Based Assessment?

A course built for skills-based assessment comes with an inventory of skills that the course developer wants their students to learn. For example, in our intro statistics course, skills include "I can represent binary predictors using treatment coding" and "I can transform a variable into z-scores"; in a more coding-focused course, skills might be something like "I can explain the difference between while loops and for loops" or "I can use pseudocode to write basic sorting algorithms".

The basic idea behind one possible variant of skills-based assessment, the one my team and I are using, goes as follows. As the course progresses, students work on developing each skill. They get multiple opportunities to show that they've gained each one. The only firm deadline is the very end of the course; until then, students can try to demonstrate a skill as many times as they need to or want to, and they won't be penalised at all for making multiple attempts. They get feedback on every attempt—why it succeeded or where it fell short. Once a student has successfully demonstrated a skill, then in the course's record-keeping, they have earned that skill permanently and cannot un-earn it. Each student's course outcome (whether number grade, letter grade, pass/fail, or something else) is some function of how many skills they earned.

This basic idea is known in different circles under different names. For example, Clark and Talbert (2023) and Buckmiller, Peters, and Kruse (2017) call it "standards-based grading"; O'Leary and Stockwell (2021), (2022), and Zuraw et al. (2019) call it "skills-based grading"; and it's also similar to a cousin framework called "specifications-based grading" (Nilson 2014; Clark and Talbert 2023).

I like the "skills-based" part of the name because it centres the skills, but I steer away from the "grading" part for a couple reasons. First, the skills-based course that my colleagues and I are developing will culminate in a pass/fail outcome, not a number/letter grade. I don't want the term "skills-based grading" to mislead students about how they'll be assessed. Second, I consider grades on the whole to be an unnecessary evil (see, e.g., Kohn 1993; Stommel 2023; Blum 2020), but "assessment" can be about assessing how a student is doing, about checking in and giving feedback on their learning journey—an approach which better encourages genuine learning (see, e.g., Ambrose et al. 2010; Sambell, McDowell, and Montgomery 2012).

Why Is the Skills-Based Framework Well-Suited for Teaching New Programmers?

In this section, I'll highlight three features of skills-based assessment that, in my view, make it much better than traditional weighted-averages grading for teaching programming to new programmers. Skills-based assessment:

1. reduces students' academic anxiety and stress;
2. lets students learn at their own pace; and
3. helps students facing structural disadvantages to succeed.

1. Reduces Students' Academic Anxiety and Stress

Many students, especially those from non-STEM disciplines, are anxious about learning to program (an observation to which a whole chapter in this volume is dedicated: Oldnall et al. 2026). And learning to program in a traditionally-graded course is like adding salt to the wound. Learning is messy, and mistakes are normal—arguably essential—along the way. But in traditionally-graded courses, mistakes are punished by lowering marks.

On the other hand, the skills-based framework acknowledges and encourages the messy process of learning. For example, allowing students the chance to demonstrate a skill repeatedly, with no penalty for re-attempts, fosters in them a sense that it's okay to make a mistake and worthwhile to try again (O'Leary and Stockwell 2021). And students also gain a sense of self-efficacy as they realise they can accomplish something that they couldn't do before (Zuraw et al. 2019). This leniency is a major reason why skills-based assessment has repeatedly been shown to lessen students' academic stress (e.g., O'Leary and Stockwell 2021; Buckmiller, Peters, and Kruse 2017; Zuraw et al. 2019; Lewis 2022).

In an ongoing research study, my colleagues and I are gathering data about how our skills-based course redesign impacts how our students feel about learning statistics and R programming. Based on the literature mentioned here, we hypothesise (and very much hope!) that our redesign will help ease our students' anxiety and stress.

2. Lets Students Learn at Their Own Pace

Programming can be hard to learn. And while some people will take to it straight away, others need more time to develop this new way of thinking (Caspersen and Bennedsen 2007; Offutt et al. 2017). But students in traditionally-graded courses don't always have the time they need. Students typically lose marks if they haven't grasped a concept by the time the course developer thinks they should have—even if they do figure it out later.

In skills-based courses, learners are not held to the course developer's timeline. Students can demonstrate skills at any point during the course, so learners who benefit from a bit more time to learn can still succeed. Offutt et al. (2017) report that letting students pace their own learning in an introductory computer science course has led to great results: students learn better and are reportedly disinclined toward academic dishonesty, because the time pressure that had motivated them to cheat has been lifted.

3. Helps Students Facing Structural Disadvantages To Succeed

The culturally ideal programmer is an able-bodied white man. Anyone who diverges from this ideal, along whichever dimension or dimensions, has extra cultural and emotional work to do when learning to code.

For example, Carter and Jenkins (1999) observed a "growing perception that female students are weaker at programming than their male counterparts" (p. 3), noting that female students tend to be less confident than male students and to underestimate their own ability more. In my earlier experience as a young female programmer, and now as a female teacher with many young female students, not much has changed about these views in the nearly three decades since.

And adding in the intersectional experience of race, Rea (2022) recounts the striking words of Olivia, who organises coding bootcamps for marginalised and minoritised people:

> I'm teaching women how to code. But it's bigger than that. What I'm really teaching them to do is how to unlearn what they've been taught. A lot of women of color, or women in general, have not been taught that science, math, coding, you name it, is for them. You're unlearning a thing, and you're learning a new skill.

Further dimensions compound the challenge of belonging in traditional academic spaces, no matter what's being taught: physical disability, mental disability, care responsibilities, work duties, a different first language than the language of education. All of these disadvantages can present a serious barrier in traditionally graded courses, in which students are commonly graded on things that go beyond the course's content-related learning objectives, such as regular attendance, active verbal in-class participation, and (as mentioned above) the timeliness of their understanding.

But in the skills-based framework, barriers like these are softened. Where clear expectations and goals are provided, less background knowledge is required, making the space more inclusive to people with different backgrounds (Parker 2018).[132] As such, skills-based courses have been shown to achieve more equitable outcomes for a wider range of learners (e.g., by O'Leary and Stockwell 2021, 2022). I really hope that the more flexible structure of skills-based assessment will help our intersectionally disadvantaged students build their skills as well as their sense of self-efficacy in a more compassionate space.

Finally, a note for even the culturally ideal student of programming: any person at any time could experience sickness, injury, or grief, or may just need time away to, say, bring their cat to the vet. Everybody benefits from inclusive pedagogy.

For further discussion of inclusion and accessibility in teaching programming, see Colquhoun et al. (2026) in this volume.

Case Study: Developing a Skills-Based Course Introducing Statistics Using R

For all the reasons discussed above and more, my team and I in Psychology at the University of Edinburgh are using the skills-based framework to redesign our undergraduate statistics curriculum. The redesigned first-year course (link to the University of Edinburgh course catalogue DRPS)[133] will launch in September 2026, so as of this writing, no students have experienced it yet. In this section, I'll describe the path we took (largely guided by Clark and Talbert 2023, chap. 11) and some key decisions we made along the way.

Our Development Process

We began by dreaming up, in very broad strokes, what topics and methods we wanted to cover in our ideal version of the course. Our early notes included, for example, "intro tidyverse", "descriptive stats", and "simple linear model". Once we all agreed on a set of topics and a rough order in which they'd appear, we dove into the weeds to develop a finer-grained list of things we want students to be able to do, such as:

- Install packages in R.
- Create new columns in a dataframe.
- Summarise numeric variables using the mean.

We ended up with a list of skills numbering between 80 and 100. This list grew and shrunk and changed over time as our ideas kept flowing. These are far from the set of skills we'll actually assess, but this early representative list of skills was absolutely good enough for the steps that would follow.

At the same time, we also considered the course's submission and feedback systems. How and when can students evidence skills? What kind of feedback will they receive?

We observed that many skills could be automatically checked (for example, identifying the code that would accomplish a given task), while a handful would require human review (for example, writing up a description of a given data set).

We decided to check most skills automatically using quizzes on our learning management system Blackboard Learn. These quizzes pull from a pre-built question pool for each skill. If students answer a question correctly, then that counts as evidencing the given skill. After submitting their answer, students will see automated feedback about how to solve the question and common misconceptions.

We decided to check the remaining skills that required human review at two points, one at the end of each semester. Students would be able to submit their responses (also on Learn) at any time, but to keep marking overhead reasonable, we only review their responses and give feedback twice a year. The feedback will be uniform across all students and simple, something like "successful", "almost", or "not yet" (Clark and Talbert 2023). For more specific guidance, students will be able to talk to us during the weekly coding workshops and our office hours.

Once these basic structures were in place, we began properly consulting with stakeholders (Sackstein 2015). From the beginning, we had support from the head of our subject area, which made every other consultation much simpler. We asked for input from the following groups of people:

- Students (but don't expect a huge turn-out—we hosted an in-person drop-in feedback session, to which two of our approximately 400 students came, and we also sent out an online feedback form, to which four students responded).

[132] A well-formulated rubric may accomplish a similar goal, but rubrics tend to have two issues that skills do not. For one, by their nature, rubrics focus on the work's deficits until the very highest levels of achievement. Here is an example from one of our earlier report-based assessments: "Results are mostly clear and logically presented. Interpretation is mostly accurate but may lack depth in linkage to analysis strategy". A rubric targets all the ways the student's product falls short from the ideal. And for a student who already feels like they don't belong, deficit-focused approaches like this will not make them feel more welcome. And for another, imagine being a student and receiving the feedback: "to get a higher mark, you need to add more depth in the linkage to analysis strategy". What does it *mean* to "add depth"? And how will you know if you've successfully done it? Rubrics tend not to be as explicitly actionable as skills, and they tend to require background academic knowledge to successfully interpret—knowledge which educators take for granted but which not all students will have.

[133] https://www.drps.ed.ac.uk/26-27/dpt/cxpsyl08013.htm

- Academic staff (we hosted a staff-focused drop-in session and sent out an online feedback form, which altogether yielded a handful more responses from staff than from students).
- Professional services staff (especially the very knowledgeable people who deal with the back-end of inputting student grades into the institution's software system; they helped us identify and solve problems we had never considered).

After integrating the very useful input we got from all these people (examples of which I'll mention below), we submitted the proposal to the committee that processes course changes.

At that point, with the administrative overhead off our desks for a while, we started to fine-tune the skills that the first iteration of the course will actually assess.

Top Tips For Writing Assessment Skills

Designing the skills themselves is a creative logistical challenge with many degrees of freedom. In this section, I'll walk through how we resolved two major questions that came up repeatedly along the way: how specific vs. how high-level should each skill be? And what level of cognitive engagement should each skill call for?

Granularity of Assessment Skills

In our early list of topics, we wrote, for example, that we want students to learn to summarise data using the mean, the median, and the mode. How many skills should we use to assess this learning?

One option is to use one skill per measure:

- I can summarise numeric variables using the mean.
- I can summarise numeric variables using the median.
- I can summarise categorical variables using the mode.

Another option is to consolidate both numeric summary measures into a single skill:

- I can summarise numeric variables using appropriate measure(s) of central tendency.
- I can summarise categorical variables using appropriate measure(s) of central tendency.

A third option is to consolidate again:

- I can summarise variables using appropriate measure(s) of central tendency.

In principle, one could consolidate arbitrarily far, depending on the focus of the course: from "I can conduct an exploratory data analysis" up to "I can analyse data", or even beyond. Here, for the purposes of illustration, I will focus on the lowest three levels, since they are the most pertinent ones for our specific aims.

The relationship between these different levels of skills is shown in Figure 30.1, along with some useful vocabulary offered by Clark and Talbert (2023, 159).

At the very top of this hierarchy, in darkest blue, are what Clark and Talbert call the "course-level objectives": the broad, encompassing learning objectives for the entire course. Our running example belongs to a course-level objective like "I can summarise data", an objective which involves measures not just of central tendency but also of dispersion, for example.

Jumping to the very bottom of this hierarchy, in lightest blue, we see the "lesson-level objectives": what we want students to learn from individual lessons. Lesson-level objectives map fairly cleanly to the fine-grained list of learning objectives that we drafted early on.

In between these two extremes, in medium blues, we see the "assessment-level objectives". Somewhere in this middle ground are the skills that we will assess.

Do we want to make absolutely sure that students can summarise numeric and categorical variables individually? If we do, then a larger quantity of lower-

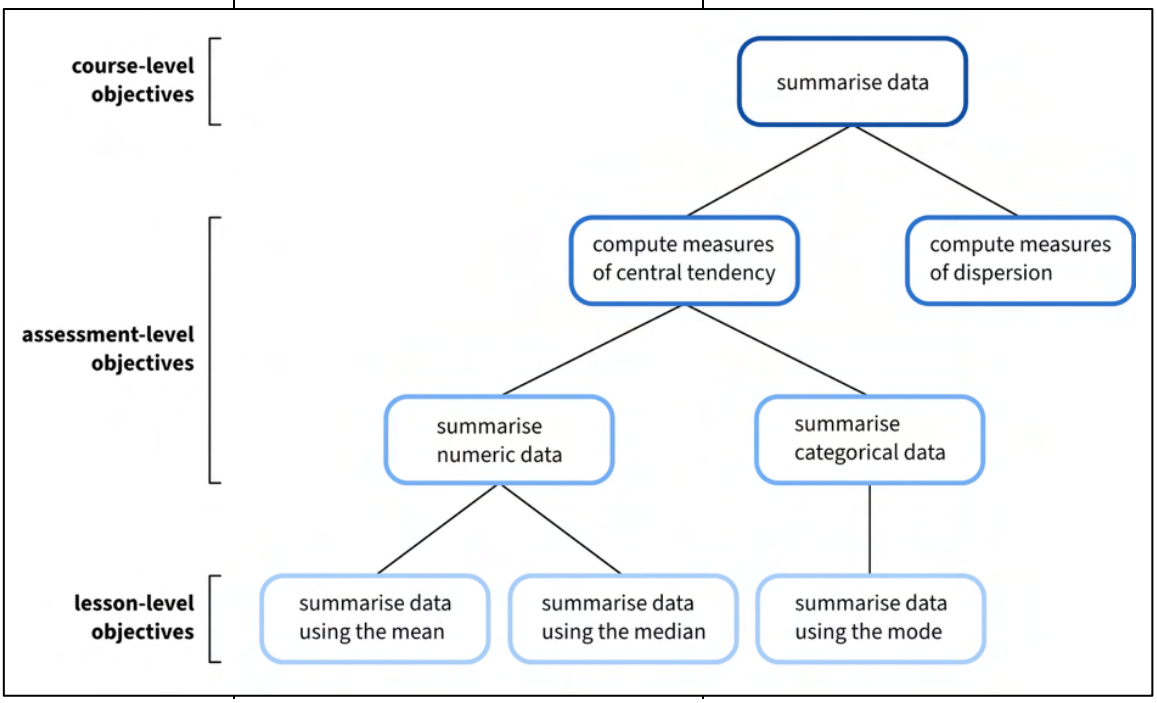

Figure 30.1: An example hierarchy of data summary skills, annotated with the levels described by Clark and Talbert (2023): course-level objectives, assessment-level objectives (a subset of which are the skills we want to assess), and lesson-level objectives.

level skills may be the way to go. Or do we assume that if a student can correctly find a sample's mean, say, they can probably also find the median and mode? If that assumption seems reasonable, then the higher-level skill might suffice.

In a coding course, an analogue to this quandary might be the following: Will students be assessed on appropriate use of control structures, broadly speaking? Or separately on being able to appropriately use if/then/else statements and various kinds of loops? Or, finer-grained, on specific structures like if/then, if/else, for loops, while loops, and so on?

In our case, for the data summary example, it is reasonable enough to assume that a student who can find the mean can probably also find the median and the mode. And further, we are also trying to keep the sheer number of skills in check (the current list, as of this writing, contains about 40). We therefore chose to assess the most consolidated, broadest, highest-level version of skills wherever we could.

Cognitive Level of Assessment Skills

Each skill should call for an action that a student can easily understand and that an assessor can easily observe (Clark and Talbert 2023). But for a given skill, we found that often, several different actions will all seem reasonable. For example, after students learn about t-tests, do we want them to be able to *conduct* a t-test, or to *explain* the logic behind a t-test?

Choosing the action verb is a weighty decision. Different verbs, like "conduct" vs. "explain", invoke different levels of cognitive engagement, as we know from frameworks like Bloom's taxonomy (Bloom et al. 1956; Anderson and Krathwohl 2001) and the SOLO (Structure of the Observed Learning Outcome) taxonomy (Biggs and Collis 1982).

In an early version of the course design, we had imagined that students could achieve one of three possible course outcomes: Merit, Pass, or Fail. This design allowed us two sets of skills: a "core" set of skills, which would cover simply conducting statistical tests, as well as an "advanced" set of skills, which would add on the deeper conceptual understanding. If students demonstrated both core and advanced skills, they would receive a "Merit" outcome; if they demonstrated only core skills, they would receive a "Pass". (I note that pedagogically, dangling the "Merit" carrot may have many of the same demotivating effects as traditional grading—see, e.g., Kohn 1993.) We discovered, though, that a categorical three-level course outcome is impossible within the strictures of our institution's software system—one example of many to illustrate why consulting with professional services colleagues is absolutely key.

In the later version of the course, we narrowed the outcomes down to Pass and Fail (an option welcomed by several students we talked to, who liked that it frees them from the stress of "chasing an A"). With no "Merit" category, we set aside the core vs. advanced sets of skills. Again, we faced a choice between "conduct" and "explain"—but again, Clark and Talbert (2023) offer guidance. They comment (p. 165) that hands-on, skills-building courses tend to use vocabulary from the more fundamental, less abstract levels of Bloom's taxonomy, whereas more conceptually-focused courses tend to use the more abstract, higher-level verbs. We want our students to focus on building skills, so we opted for the more hands-on "conduct a t-test" variant throughout.

The Big Skills-Based Picture

To conclude, let's zoom back out: what makes skills-based assessment well-suited for helping students learn to program? The picture I hope to have painted in this chapter shows a different way of checking in on how people learn, one that genuinely recognises students as the learners they are, not just as faulty repositories for the information we bestow (Freire 1996). Because skills-based assessment mitigates common causes of anxiety and softens many structural barriers that students face, it is our framework of choice for helping new programmers start their learning journey.

Acknowledgements

This chapter would not exist without:

- Franziska McManus, who first suggested I share these ideas in this book.
- Itamar Kastner, who introduced me to skills-based assessment and who continues to inspire the best parts of my practice.
- Emma Waterston, Josiah King, and Umberto Noè, my colleagues on the Psychology stats team, who are all incredibly caring educators and fun, thoughtful, and thought-provoking collaborators.

References

Ambrose, Susan A, Michael B Bridges, Michele DiPietro, Marsha C Lovett, and Marie K Norman. 2010. *How Learning Works*. San Francisco: Jossey-Bass.

Anderson, L. W., and D. R. Krathwohl. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman.

Biggs, John B., and Kevin F. Collis. 1982. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, Inc.

Bloom, Benjamin, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals*. Handbook 1: Cognitive Domain. London: Longman.

Blum, Susan D., ed. 2020. *Ungrading: Why Rating Students Undermines Learning (and What to Do Instead)*. West Virginia University Press.

Buckmiller, Tom, Randal Peters, and Jerrid Kruse. 2017. "Questioning Points and Percentages: Standards-Based Grading (SBG) in Higher Education." *College Teaching* 65 (4): 151–57. https://doi.org/10.1080/87567555.2017.1302919.

Carter, Janet, and Tony Jenkins. 1999. "Gender and Programming: What's Going On?" In *Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education*. https://doi.org/10.1145/305786.305824.

Caspersen, Michael E., and Jens Bennedsen. 2007. "Instructional Design of a Programming Course: A Learning Theoretic Approach." In *Proceedings of the Third International Workshop on Computing Education Research*, 111–22. Atlanta Georgia USA: ACM. https://doi.org/10.1145/1288580.1288595.

Clark, David, and Robert Talbert. 2023. *Grading for Growth: A Guide to Alternative Grading Practices That Promote Authentic Student Learning and Student Engagement in Higher Education*. Routledge.

Colquhoun, Rebecca, Samantha Ahern, Brittany Blankinship, and Patricia Loto. 2026. "Practices to Foster Inclusion and Accessibility in Programming Teaching." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Freire, Paulo. 1996. *Pedagogy of the Oppressed*. Translated by Myra Bergman Ramos. Penguin.

Kohn, Alfie. 1993. *Punished by Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes*. Boston: Houghton, Mifflin and Company.

Lewis, Drew. 2022. "Impacts of Standards-Based Grading on Students' Mindset and Test Anxiety." *The Journal of Scholarship of Teaching and Learning* 22 (2). https://doi.org/10.14434/josotl.v22i2.31308.

Nilson, Linda B. 2014. *Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time*. Sterling, VA: Stylus.

O'Leary, Maura, and Richard Stockwell. 2021. "Skills-Based Grading: A Novel Approach to Teaching Formal Semantics." *Proceedings of the Linguistic Society of America* 6 (1): 869. https://doi.org/10.3765/PLSA.V6I1.5025.

———. 2022. "Implementing Skills-Based Grading in a Linguistics Course." *American Speech* 97 (2): 247–62. https://doi.org/10.1215/00031283-9940629.

Offutt, Jeff, P. Ammann, K. Dobolyi, Christian Kauffmann, Jaime Lester, Upsorn Praphamontripong, H. Rangwala, Sanjeev Setia, Pearl Y. Wang, and Lisa N. White. 2017. "A Novel Self-Paced Model for Teaching Programming." In *ACM Conference on Learning Scale*, 177–80. ACM. https://doi.org/10.1145/3051457.3053978.

Oldnall, Chris, William Kay, Chris Sutherland, Tiago A. Marques, and Robert Young. 2026. "Overcoming Coding Anxiety: Lowering the Stakes and Making It Fun." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Parker, Priya. 2018. *The Art of Gathering*. Penguin.

Rea, Ashley. 2022. "Coding Equity: Social Justice and Computer Programming Literacy Education." *IEEE Transactions on Professional Communication* 65 (1): 87–103. https://doi.org/10.1109/tpc.2022.3143965.

Sackstein, Starr. 2015. *Hacking Assessment: 10 Ways to Go Gradeless in a Traditional Grades School*. Vol. 3. Hack Learning. Times 10 Publications.

Sambell, Kay, Liz McDowell, and Catherine Montgomery. 2012. *Assessment for Learning in Higher Education*. Routledge.

Stommel, Jesse. 2023. *Undoing the Grade: Why We Grade, and How to Stop*. Hybrid Pedagogy Inc.

Zuraw, Kie, Ann M. Aly, Isabelle Lin, and Adam J. Royer. 2019. "Gotta Catch 'Em All: Skills Grading in Undergraduate Linguistics." *Language* 95 (4): e406–27. https://doi.org/10.1353/lan.2019.0081.