Assessment and Marking in an Introductory Programming Course for Physics Students

Ben Waugh

Department of Physics and Astronomy, UCL
b.waugh@ucl.ac.uk

Rebecca Chislett

Department of Physics and Astronomy, UCL

Louise Dash

Department of Physics and Astronomy, UCL

This informal case study describes how assessment is implemented in an introductory programming course for first-year undergraduates studying physics. It is not offered as a model to follow, but we hope someone developing a similar course (not necessarily in the same domain) will learn something from the approaches we have used, and how we have adapted them over time.

Syllabus and Course Design

Teaching Computing Alongside Practical Skills and Data Analysis

PHAS0007 Computing is one of three components of a 15-credit module on *Practical Physics and Computing* taken by all undergraduates in the Department of Physics and Astronomy at UCL in the first term of their first year, the other components being *Experimental Laboratory*, and *Data Analysis and Statistics*. After this introductory module, students take different computing courses depending on whether they are registered for a degree in *Physics*, *Astrophysics*, or *Theoretical Physics*.

The three components are taught separately, but the syllabus is aligned so that students can use computing to carry out data analysis, and apply both skill sets to the results of their laboratory experiments. However, the computing component is not just about statistical analysis: we also introduce basic computational physics, and teach students more general skills in building complex logic incrementally, structuring and commenting code clearly, and creating a scientific report in the form of a computational notebook.

Computing Classes

In the academic year 2025-26 we had over 300 students, divided into five groups because of the limited laboratory space, with each group having one 3.5-hour experimental laboratory session and one 3.5-hour computing class in each of the ten teaching weeks in the term. All students also have a weekly lecture on data analysis and statistics. Computing sessions take place in one of our experimental laboratories, with an individual Windows PC available for each student, although some choose to use their own laptop computer instead.

What We Teach

Our aim is not to teach coding as a general skill, or computer science as a discipline, but to enable our students to apply computational techniques to their work as physicists. We assume no prior knowledge of programming: while 87% of the 190 respondents to our pre-course survey said they had some experience of Python, 38% had not been taught programming at all and a further 21% “very little”. We use the Jupyter[137] Notebook application throughout the course, rather than an integrated development environment (IDE) or the more sophisticated but complex JupyterLab interface. We limit the extent of the computing concepts we attempt to teach in the course, using only simple data structures (lists and NumPy arrays but not dictionaries) and teaching students to define functions but not classes. We start with arithmetic operations, data types and variables, and importing and using existing functions. In the second week we introduce lists, arrays, loops, and defining functions. Later units cover plotting data, linear regression, summing series, and basic computational physics.

[137] https://jupyter.org/

Assessment Framework and Principles

The module has six summative assessment elements, a large number but one that reflects the wide range of skills that are covered in the three components. There are two assessments within the practical course (a formal report and a notebook assessment contributing 25% and 15% respectively to the module grade), one multiple-choice quiz on data analysis and statistics (contributing 15%), and three further elements within the computing component. Two of the computing assessments are extended "take-home" assignments: one mid-term data analysis assignment testing the concepts students learn in the statistics component as well as their programming skills, and a final computational physics task, contributing respectively 15% and 20% of the module grade. The remaining 10% comes from assessed weekly coursework exercises throughout the term.

Our approach can be described as *assessment for learning* (Wiliam 2011) in that "*all* assessment, within the overall package, should contribute to helping students to learn and to succeed" (Sambell, McDowell, and Montgomery 2012). Rather than creating separate learning activities and subsequent assessed tasks to measure performance, we try to design exercises that are interesting and develop understanding while also forming the basis of students' grades for the course.

UCL's regulations require the mark for each assessment component to be expressed as a percentage, and the overall module mark to be calculated as a weighted average of component marks. Within each component, module leaders can decide on how marks are decided, but all summative assessment must be *criterion-referenced*: marks are decided by checking submitted work against defined standards and not against the rest of the student cohort.

Continuous Assessment

Integrating Assessment With Learning in Weekly Sessions

The core teaching materials are a series of Jupyter notebooks that explain and illustrate the concepts we want students to understand, include code examples, and culminate in a programming task that guides students through the application of the week's material. Following consistent requests from students we also provide recorded screencasts talking through the content of the notes. In the class sessions, academics and postgraduate teaching assistants (PGTAs) offer individual help with understanding the examples and tackling the assignment. In 2025-26 each session was staffed by one of the two course leaders along with two further academics and two PGTAs. We try to guide students through the process of debugging their own problems rather than simply offering answers.

There are nine weekly *units* in the course: after the first five comes the UCL reading week, during which students work on the mid-term data analysis assignment, as well as assessments for other modules, but have no lectures or practical sessions. Then come five more sessions, although the last of these is not a unit in its own right but an opportunity for students to get help on the first stages of the final assignment and to catch up on any missed work.

The key aim of these tasks is formative: we explicitly encourage students to focus on the feedback they receive rather than on where they have "lost marks". Each unit contributes only around 1.1% (one ninth of 10%) to the module grade, but we have noticed that activities without any summative aspect tend to be skipped or at least rushed through by students. Some level of extrinsic motivation can be productive (Lin, McKeachie, and Kim 2003) despite the risk of over-assessment and an "assessment arms race" (Harland et al. 2015).

Automatic Marking of Multiple-Choice Questions (MCQs)

It is not practical to cover all taught material in authentic assessed tasks because of the excessive workload this would put on both students and markers. Each unit therefore also includes an online multiple-choice quiz, marked automatically, to help students check their understanding of the material in the notes before applying it. In the past this score contributed to the grade for the unit, along with the weekly programming assignment. In order to encourage students to use the quiz as a source of rapid feedback, we allowed unlimited retakes, with the highest score achieved being used. This still added complexity without any clear benefit, so we no longer use the quiz score in calculating the unit grade, but each student has to achieve a pass mark (typically 7 out of 10 questions) before they are able to submit the main assignment for the week.

Marking Programming Tasks In-Person

Prompt and Effective Feedback

This work is graded in person in class, primarily because we find this an effective way to give feedback: what we say to each student can be tailored to their own work and forms a dialogue which the student steers by their own enquiries and their responses to our questions. In contrast, written feedback is a one-way communication channel that takes significant staff time to impart guidance that may not even be read by the student, let alone meet their individual needs. This feedback is also timely: we encourage students to submit their work at least 30 minutes before the end of the session so that they can get feedback and a grade before they leave the room. This is not always possible, and some flexibility is important for students who may for

example miss a session due to illness, so we allow submission up to the day before the following session, and grade work in the following session or as soon as possible thereafter.

Integrity and "AI Proof" Assessment

An additional benefit of in-person marking is that we can check who we are talking to, and to some extent who has done the work. This is not the case for out-of-class marking of uploaded material. Integrity has always been a concern, but has become more of a talking point recently with the expansion of generative AI tools and the ensuing search for "AI proof" assessments. This year we have added an element to the rubric for each assignment that assesses the student's ability to explain their work or to make changes to their code when asked. This has proved useful in diagnosing student misconceptions and identifying areas where more practice is needed, as well as deterring misconduct. It has, inevitably, increased the time taken to mark each piece of work, so one of our aims for next year is to reduce the time spent on the aspects of marking that are more mechanical and less interactive. This might involve the use of automatic marking, based on unit tests or the Moodle CodeRunner[138] plug-in, or simply expanding the use of MCQs.

Organization

All course staff in the lab are responsible for both helping and marking students, with the balance shifting from the former to the latter over the course of the session. Staff move around the room and sit or stand next to the student whose work they are marking, using a tablet or laptop to assign marks in an online rubric in Moodle, our learning management system. We aim to mark students in the order that they have submitted, leaving them free to continue with further investigation or other work while they wait, and use a low-tech system to manage this: once they have submitted their completed notebook on Moodle, students are asked to write their name and desk number on a sign-up sheet at the front of the room. The marker crosses the next name off the list before going to the desk, checking the identity of the student, and then reviewing their work. Numerical marks are assigned by selecting a box for each criterion in the rubric, while feedback is given verbally. The process takes from around five minutes to fifteen or more, depending on how much discussion and explanation is needed. While this is time-consuming, we believe this is a productive way to use the limited contact time available and the expertise of the staff.

Assessed Computational Notebooks

Authentic Tasks

The two longer assignments are designed to be fairly authentic tasks in that they require the creation of a computational notebook including explanatory text and structured as a scientific report rather than simply a Python script. Students have more time to work on these, so the tasks are more ambitious.

Rubric Design

In any but the most simplistic rubric, there is a need for detailed explanation of each criterion and how to decide on the appropriate level for a given student submission (Brookhart 2018). It is often useful to illustrate commonly occurring responses or misconceptions, going beyond the level of detail that is useful for any given student, or fits into the rubric format used by Moodle. Hence we keep the criterion and level descriptions concise in the rubric itself, and provide markers with detailed marking notes explaining how they should be interpreted.

One change that we believe has been helpful, and has been positively received by markers, was to reduce the granularity of marking each criterion. Instead of giving a mark out of ten, we now define levels from zero (for no or minimal work) to a maximum of three or four. The intention, albeit it not achieved everywhere, is to define criteria for each level that build on those for the level below, so only work that meets the requirements for level two is then eligible to be considered for level three. This is intended to escape the sort of ambiguity that is common in mark schemes that make use of the word *but*, as in *has a well written conclusion but inadequate reference*s, leaving the marker to decide how to deal with a report that has outstanding referencing but a mediocre conclusions. This does take away the freedom of markers to use intermediate marks to reflect this uncertainty, but we hope this is outweighed by the benefit of reducing the time spent agonizing over small distinctions that make little difference to the final grade.

In practice each level is treated as a corresponding number of points, and the overall mark is calculated as the percentage of possible points awarded. A more sophisticated approach to weighting criteria and levels would be possible, but we see simplicity as a virtue, and additional complexity as unjustified in this case.

The marking takes time, with typically around 300 lengthy notebooks submitted. These are divided across ten or so markers, but it can take on the order of 30 minutes to mark each one, with the first few for each marker taking substantially longer. Limiting this workload, and ensuring the marking is completed within the four-week institutional deadline, are areas where we are continuing to focus effort.

Marking Fairly and Consistently

Marking is not conducted in person and is done anonymously in line with UCL's academic regulations, which does mean that there is no way to be sure whether a student has completed their own work, and what help (from other students, people outside the course, or

[138] https://coderunner.org.nz/

AI software) they have received. We use JPlag[139] to look for signs of collusion, but any indications of misconduct are investigated by the course leaders and the few cases that we pursue are decided by us using our own judgement, not by any automated process. JPlag has been successful in identifying cases where two students have submitted identical or very similar sections of code or text, but is not capable of flagging AI-generated work, where even the same tool can generate different output when applied multiple times to the same input.

While distributing the marking across a pool of markers limits the workload on any one staff member, it makes it challenging to ensure fairness (McConlogue 2020, chap. 6). The course leaders are required to second-mark a representative sample of work and take measures if discrepancies are found. However, it is hard to resolve problems after the marking is complete, short of re-marking from scratch, so we have put a lot of work into supporting consistency early in the process. The design of the rubric has been adjusted each year, trying to find a balance between well-defined and "objective" criteria, which leave little space for flexibility and professional judgement, and broader, more holistic criteria. The former tend to lead to a rubric with a large number of marking decisions to be made, albeit with each being more mechanistic and faster to process, while the latter mean fewer decisions but leave a lot of scope for different interpretations to affect marks unevenly.

Training and Calibration

No document can solve this problem and make marking *reliable* in the sense of giving consistent results across markers and students, except by making the assessment less *valid* by forcing us to use criteria that are artificially constructed and don't reflect the learning we care about. We can create a better process by introducing *calibration* sessions (McConlogue 2020, chap. 6) where we discuss our approach and ideally specific examples with the pool of markers. In a university context this is difficult because our markers have other commitments and are not all available at the same time for such a session. Hence we need to arrange multiple sessions and individual discussions, limiting the extent to which each marker can learn from all the others. An approach we currently use is for one of the course leaders to mark one or more student submissions, talking through their thought process as they do so, and responding to questions from other markers. This is particularly productive when the other markers are able and willing to mark one of their own allocated submissions in parallel, and raise points for discussion.

A Structured Data Analysis Assignment

The mid-term assignment asks students to carry out a data analysis task on a physics data set. It requires them to apply the concepts they have been taught in the data analysis and statistics course, as well as using Python code, calling appropriate library functions, to fit a straight line to the data and to interpret their results. This involves creating a report in the form of a Jupyter notebook, discussing the techniques they use. The instructions for this task are quite prescriptive, specifying the tasks to be undertaken and in what order, and giving specific questions that should be discussed in the report, such as to what extent the data are well described by a linear model.

An Open-Ended Computational Physics Assignment

The final assignment has a similar format to the mid-term assessment, albeit focussed on a computational physics problem rather than data analysis. Students are again required to investigate a problem and present their results and conclusions in the form of a Jupyter notebook. The most significant difference is that the final part of the task is fairly open-ended: students are led through the initial steps of an investigation, e.g. using Euler's method to predict the motion of a planet around a star, but it is up to them how they extend this, e.g. by tracking multiple bodies, investigating more complex orbits, or applying alternative integration algorithms.

The rubric is inevitably longer because the task is more extended and divided into multiple sections. Criteria like *correct code* are treated separately for each part of the task so that it is possible to deal consistently with cases where, for example, a student has successfully completed one section but not the rest of the assignment. As a compromise between explicitness and simplicity, we do apply some criteria (e.g. *code quality: comments*) to the notebook as a whole, and accept that in these cases a student can score well even with incomplete work.

The open-ended final investigation cannot be broken down into well-defined specific criteria, so we use only two: *thorough investigation* reflecting the ambition of the work attempted, along with its correctness, and *demonstration and discussion*, assessing how well the conclusions are explained and backed up by appropriate presentation of plots or animations. These criteria are marked out of four, and specific examples are discussed among the markers in an online forum in order to build a shared understanding of how to judge these.

[139] JPlag is an open-source sytem for source code plagiarism detection (Prechelt, Malpohl, and Philippsen 2000).

The Future

We continue to make minor adjustments to the course syllabus and assessment as issues arise, with an onoging focus on providing constructive feedback and fair marks to students while limiting the resulting staff workload. Possible future developments include more explicit use of frameworks such as standards-based grading (Pankratz 2026; Clark and Talbert 2023, chap. 5) or specifications grading (Clark and Talbert 2023, chap. 6).

Larger changes are also in the offing, both locally and in the wider world. A UCL-wide *programme excellence project* offers an opportunity to merge the existing module with others to form a more coherent 30-credit module taught over the full academic year. The worlds of education, research, and software development are also seeing major changes as generative AI techniques and tools become more powerful and pervasive. This has implications for the teaching of programming across disciplines that go well beyond the scope of this article but are discussed elsewhere in this volume (Evkaya et al. 2026).

References

Brookhart, Susan M. 2018. "Appropriate Criteria: Key to Effective Rubrics." *Frontiers in Education* 3. https://doi.org/10.3389/feduc.2018.00022.

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.

Evkaya, Ozan, Hebatallah Shoukry, TJ Elmas, Laila Dabab Nahas, and Steven Watterson. 2026. "An Optimistic Outlook on Teaching, Learning and Assessment for Coding with the Emergence of Generative AI." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Harland, Tony, Angela McLean, Rob Wass, Ellen Miller, and Kwong Nui Sim. 2015. "An Assessment Arms Race and Its Fallout: High-Stakes Grading and the Case for Slow Scholarship." *Assessment & Evaluation in Higher Education* 40 (4): 528–41. https://doi.org/10.1080/02602938.2014.931927.

Lin, Yi-Guang, Wilbert J. McKeachie, and Yung Che Kim. 2003. "College Student Intrinsic and/or Extrinsic Motivation and Learning." *Learning and Individual Differences* 13 (3): 251–58. https://doi.org/10.1016/S1041-6080(02)00092-4.

McConlogue, T. 2020. *Assessment and Feedback in Higher Education: A Guide for Teachers*. UCL Press.

Pankratz, Elizabeth. 2026. "Developing a Skills-Based Framework for Assessments in Programming Courses." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Prechelt, Lutz, Guido Malpohl, and Michael Philippsen. 2000. "JPlag: Finding Plagiarisms Among a Set of Programs." 1. Vol. 2000. Interner Bericht. Fakultät Für Informatik, Universität Karlsruhe. https://doi.org/10.5445/IR/542000.

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

Wiliam, Dylan. 2011. "What Is Assessment for Learning?" *Studies in Educational Evaluation*, Assessment for Learning, 37 (1): 3–14. https://doi.org/10.1016/j.stueduc.2011.03.001.