Leveraging Jupyter Notebooks in Assessment Development, Completion and Marking to Reduce Cognitive Load and Minimise Errors

Oli Howson
The Open University, United Kingdom
oli.howson@open.ac.uk

Abstract
When redeveloping a level two Algorithms and Data Structures module the decision was made to centralise the learning material and assessments around Jupyter Notebooks. We shall explore the lessons learnt and advantages gained through an assessment process which now has development, assessment and marking all completed within the Jupyter Notebook infrastructure.

The cognitive load associated with assessment development often stems from the complexity of the task and the risk of introducing errors during various stages of the process. By leveraging Jupyter Notebooks, authors can benefit from a range of features that enhance productivity, promote collaboration, and reduce the likelihood of errors.

This presentation will delve into several key areas where Jupyter Notebooks can significantly contribute to the assessment development process. We will discuss the seamless integration of code, documentation, and visualisation capabilities, which allows developers to write and test their assessment in a single environment. This not only enhances readability and maintainability but also facilitates the identification and rectification of errors through an interactive and iterative development process.

We will explore the collaborative nature of Jupyter Notebooks within the GitHub infrastructure, enabling developers and reviewers to work together. Furthermore we will explore the in-house development of a plug-in to reduce the chances of amendments becoming desynchronised between student-facing and assessor-facing versions.

Jupyter Notebooks can also help reduce cognitive load for students who are taking assessments. Traditional assessments can often be challenging to follow, with a series of complex instructions, questions, and response formats. Even digitally completed programming assessments may have question documents, answer documents and multiple program code files to be read, changed and submitted. This can lead to cognitive overload, especially for students who may already be feeling stressed or anxious about the assessment.

Jupyter Notebooks can help mitigate this by providing an interactive and visually appealing assessment experience. By presenting questions, code and responses in a single digital format, Jupyter Notebooks allow for more flexible and intuitive navigation, reducing the cognitive load required to understand and respond to questions.

Additionally, Jupyter Notebooks can provide immediate feedback to students, providing a sense of clarity and direction. By offering automated testing and visual cues, such as colour-coded responses, Jupyter Notebooks can help students quickly identify areas to be developed and adjust their approach accordingly.

The final area of utilisation of Jupiter Notebooks is in the assessment process; markers are provided a single document per student to mark, without having to view additional code files. As well as again reducing cognitive load, assessors are saved the potential hassle of having to chase students for additional files which should have been submitted but may be missed. We will look at a second in-house plugin which has been developed to aid and streamline the assessment process.

Keywords: Assessment, Distance Education, Programming, Cognitive Load, Programming

1. Introduction

M269 (The Open University, 2023) is a second-level (year two) degree module focussing on algorithms, data structures and computability. While it utilises Python programming heavily, it is not a programming module; rather programming is used to illustrate and exemplify the Computer Science learning points within the module. Traditionally M269 has been delivered in what, for the Open University, is a traditional manner; presented over nine months with multiple auto-marked formative assessments, two tutor-marked assessments (TMAs) at intervals roughly 1/3 and 2/3 through the course, and a final examiner-marked assessment (EMA). The TMAs involved answering short and medium length answers and writing significant portions of Python code into existing Python files. Questions were on a question paper, with answers either being inserted into the question paper, a separate answer document, or within the Python files. Students had to submit the correct answer document and all of the relevant Python files. The EMA was paper based, consisting of short, medium and long written answers and were either hand-written (in an examination hall) or, during COVID, typewritten into a separate word-processed document.

A number of problems occurred during the writing, administration, completion and marking of these assessments. These are discussed here, along with the iterative attempts made to solve or reduce the impact of the problems by introducing Jupyter Notebook based assessments.

1.1 Versioning Differences

During development, authors shared versions of the assessments with one another for critical reading and review. As each assessment consisted of numerous files, this often led to oversights where files were missed, and differing versions were reviewed. With no formally agreed template, questions often looked quite different as well, in both format and style.

TMAs are made available some time before the submission date, and invariably colleagues or students would spot errors in the documents that would need clarification or amending. It quite frequently occurred that the corrections would be made in one version of the files but not others, meaning that the files that students and their tutors were looking at did not always marry up.

1.2 Completion Issues

The real problems came in the completion of the assessments by the students – particularly in the TMAs which had multiple Python files, data files, and often differing question and answer papers. Having to move between so many documents ran the chance of increasing cognitive load on the students unfairly as students must mentally switch between different resources, locate relevant information, and maintain a clear understanding of the overall assessment structure. The context switching and transitioning between different task concepts can disrupt the flow of thinking significantly. Supporting library code files, test harnesses, students' own code files and the like represent extraneous cognitive load while the question content itself represents intrinsic cognitive load (Cerdan, Candel, & Leppink, 2018). It has been suggested that when only minimal extraneous

cognitive load is required by the student, they can cope with a higher intrinsic cognitive load which can in turn result in more learning or, in our case, better performance under assessment (Lafleur, Côté, & Leppink, 2015).

1.3 Marking Issues

Further problems occurred when colleagues attempted to mark the submitted assessments. First of all, there were a number of files to be submitted; as well as the answer document there were often numerous python files each of which represented an individual question, part or sub-part. Where a file was missing there could be significant marks lost. And as the files were potentially dependent a single mistake in one could cause a knock-on impact which could, unfairly, cost students further marks. Furthermore, due to student submitting sometimes relatively large numbers of extraneous files such as duplications of data and library files, temporary files and so on, the submission could be quite time consuming for the tutor to navigate.

The cognitive load could impact the tutors as much or more than the students. They also had to jump between multiple files just like the students. However, many tutors like marking question by question rather than student by student. While this is simple with printed assessments, with many files in many folders and sub-folders it really wasn't possible in the existing situation.

Tutors also had to complete feedback in differing ways depending upon the question type, with a mix of comments on the word or pdf answer document and embedded comments within the Python code. As providing feedback can already be one of the most time-consuming aspects of marking (Cavanaugh & Song, 2014)so any additional hurdles can cause significant frustration.

Finally, some programming questions were able to be automatically tested by running tutor code. This meant the code files had to be copied to the student folder(s), the student filenames, function names and so on checked so that the tutor code could find them, and then the test code loaded into the development environment and run to check the results.

2. Module Rewrite

When the original module came around for a rewrite, the production module chair decided to craft the whole module in Jupyter Notebooks – the original module had many web pages on the VLE (Virtual Learning Environment), a printed textbook, and downloadable Python files. Jupyter Notebooks allowed students to explore, write and execute code in an interactive manner. This means they can run code snippets and see their outputs immediately. With the more traditional method, students would typically have to type out and execute code in a separate environment and switch back and forth to view results. Jupyter Notebooks have been shown to help develop interdisciplinary problem solving and resilience (Willis, Charlton, & Hirst, 2020).

Jupyter Notebooks provide a rich environment where we could combine code, text, visualizations, and multimedia elements like images and videos. This makes it easy for the author to explain their thought process and create interactive narratives. It's a powerful tool for educational purposes. In the more traditional approach, it is harder to maintain a coherent narrative while moving between book, VLE and code files (Perez & Granger, 2021).

Jupyter Notebooks provide seamless integration with data visualisation libraries like Matplotlib and NetworkX which have been used within the material, amongst other things, for demonstrating graphs. We could generate visualisations and immediately display them within the Notebook; students could adapt and change the module code and re-display the visualisations, allowing for a more interactive and exploratory code analysis process. While it's possible to display visualizations on the VLE webpages, they would have been static, relying

on students to install additional software (of the correct versions) in their development environment if they wanted to explore that code themselves.

The production chair wrote supporting scripts to convert the markdown the content was written in into Jupyter Notebooks and thence to HTML and PDF versions automatically (as well as producing separate Python files) to support accessibility issues. Jupyter Notebook is not currently considered web-accessible (Al-Gahmi, Zhang, & Valle, 2022), while the HTML and PDF versions have all of the accessibility tools available with those formats.

Moving to Jupyter Notebooks had another advantage; they enabled the development team to have complete control over the content, without having to "make do" with the textbook author's work or jump through administrative, and time-consuming, hoops to have the content put on the VLE. Changes and improvements to the material can be released to students very quickly.

3. Assessment development

With the learning material being developed in Jupyter Notebooks, it felt counter-intelligent to continue producing assessment material using multiple Word/PDF files and Python code files; with the assessment model changing to three tutor marked assessments (TMAs) and an "examiner marked TMA (emTMA)" replacing the more traditional exam, the decision was taken to produce the assessments in Jupyter Notebooks as well, with all of the affordances offered above being available to the assessments.

3.1 General approach

There were several general decisions made by the module team; firstly, the intent was to have the Jupyter Notebook "front end" the assessments, so these were the only documents students needed to use and edit. Initially, as each TMA is completed in two halves (for example part 1 in week 5, part 2 in week 10) the TMAs were split into two Jupyter Notebooks. As content was reduced slightly for the second presentation of the module each TMA has now been combined into a single Jupyter Notebook file for the second presentation. Each TMA is out of 100 marks. There is the intention to use automatic marking as much as possible. This enables students to automatically trial their code with sample test data for an indication of wrongness. The sample test data is insufficient for them to guarantee rightness unless they expand it to include all scenarios. Additionally, the tutors will be able to automatically test their code with unique unseen test data which should cover all scenarios and therefore indicate correctness. There is a test harness built into the production material which is reutilised for this purpose. Auto grading has been shown to improve the student experience by reducing feedback time and removing bias in scoring (Hahn, Navarro, Burgos, & De Lan Fuente Valentin, 2021). Furthermore, as the test harness and test data is made available to students it has the added advantage of supporting students in testing their own code, which reduces mistakes, identifies areas for improvement and has a substantial impact on learning (Nutbrown, Higgins, & Beesley, 2016).

Each "main question", denoted by number, consists of a single topic area or scenario. This is then split into question parts (denoted by letters) and sub-parts (denoted by roman numerals). Each question, part or sub-part has a *single* area for students to write their answer; if they are required to write both code and a written solution these are split into sub-parts.

Other decisions had to be made and reinforced with the new assessment system. Unlike the previous module which had an unseen examination, students had full access to the Internet and a significant period of time to tackle the tasks. We therefore had to drop relatively simple recall questions in favour of testing understanding, to avoid students simply searching online for definitions (Davies, et al., 2022).

Furthermore, we have made a concerted effort to avoid "double jeopardy", wherein a problem with one answer leads to an inability to score well in further answers. As a rule, if a student is required to design an algorithm in Q1(a), then they will be asked to write a *different* program to *our* design in Q1(b) and analyse the complexity of a *third* algorithm or program, which we will give them, in Q1(c).

As the finished assessments became relatively long documents (with no page delineation as would be found in Word or PDF versions), we developed a JavaScript snippet which students and tutors run that automatically colour codes certain aspects of the document, based on the colours utilised by the *empinken* plugin (Hirst, 2022) used by colleagues in a later module. Areas where students need to write an answer are given a pale-yellow background. Marking guidance are given a pink background, and feedback is given a blue background.

3.2 First Iteration

The first presentation of the redeveloped M269 was run from October 2021 to July 2022 and given the designation 21J. An initial authoring and marking solution was developed shortly prior to, and within the early months of, this presentation.

3.2.1 Authoring System

As a module chair, one of the most frustrating aspects of the original assessment system was differing versions of the assessment and marking guidance floating around between students, tutors and the module team. And early decision was to create a single master document which would then be split pragmatically into student-facing and tutor-facing versions. The first iteration had code written in markdown and then sent through a Python script to generate three versions. This enabled Python to add in the specific things which were not easily accessible within Jupyter (or could be easily mis-entered), such as metadata. The three generated files were:

- A Master version containing everything; questions, answer areas, the marking guidance and areas for the tutor to provide feedback.
- A Student version with the feedback and marking guidance removed leaving just the questions and the yellow answer sections.
- A Tutor version having the answer spaces and feedback spaces removed, leaving just the questions and the pink marking guidance.

All questions, instructions and code that students need to edit are included within the Jupyter Notebook itself which is distributed within a zip file. While supporting Python files, data files, and embedded images and videos may be distributed within the zip file, these are never edited or examined by the students and therefore if students forget to return them, then they are easily replaceable by tutors. In fact, some tutors replace them as a matter of course to ensure students haven't been tempted to alter the test harness to give falsely correct results when tutors run their test code.

3.2.2 Marking System

Similarly to the production script, a Python script was shared with tutors which would go through a student's submitted work and identify answer areas. Below this it then inserted a feedback cell and a cell which prompted the tutor to enter the marks for that question part. When the Jupyter Notebook was then opened by the tutor the JavaScript snippet would colour code these sections blue. Initially the marking guidance was also copied in; however University guidelines is that marking guidance is not shared directly with students, so the script was modified to remove this. As this was a batch process, tutors could process all students in one go ready to be marked.

Tutors were then able to open Jupyter Notebook, and open each students' answer in a new tab. With the marking guidance document open either in another tab or, frequently, on a separate screen, tutors were able to very swiftly mark on a question-by-question basis (Suoto & Nadas, 2007).

3.2.3 Teething troubles

Writing in markdown is not quite as simple as writing in Jupyter. While there are systems for automatically switching between Jupyter and markdown files, some colleagues struggled with these. Writing in markdown required authors to stick to quite rigid rules, with differing heading levels indicating differing things and increasingly complex codified instructions to tell the generation script whether to create a text or a code cell. This became even more complex when a question switched between text, code, and back again. This invariably meant the author had to go through and make innumerable manual edits to the markdown to get the script to run and produce a usable Jupyter Notebook file. The interface was also somewhat unwieldy, with template files, question part files and so on in a number of folders.

The marking preparation script also had a number of teething troubles. Part of this was the software, being in active and rapid development, did not have the best user interface being entirely command line based and a little nuanced. There was also the problem that any additional cells entered by the user would not have the correct markdown flags, meaning that there was a chance that answer cells were put in the wrong place or not inserted at all. A larger problem lay in a very minor development error which "locked" student answers – ostensibly to stop them being mistakenly edited by the tutor. The outcome of this was that tutors were also unable to run the code cells or make direct comments, leading to a somewhat onerous manual process to unlock them.

That being said, marking went relatively smoothly. While there was some concern by the marking team initially, a lot of the worries never actually came to pass; some of the more technically minded tutors tweaked the marking script and fed this back to the module team and their colleagues which led to further incremental improvements.

3.3 Second Iteration

In 2022-23 the second presentation of the new module material, and hence the new module assessments, was run. The Python scripts were replaced with plugins to smooth the workflow for the assessment authors and the tutors that had to mark the assessments. Jupyter Notebooks are extensible, with the nbextension (Jupyter Notebook Extensions, 2023) toolkit frequently being used to create plugins for underlying infrastructure. This was used as a starting point for the second iteration, although a lot of the functionality interacted directly with the Jupyter Notebook itself.

3.3.1 Writing Plugin

The authoring plugin allows the assessment authors to work directly within Jupyter Notebooks, creating the master file. As this is a Jupyter Notebook already, it needs no conversion (and hence has fewer chances of going wrong). The plugin is driven from a single button on the toolbar which brings up a dialogue box giving the author a range of options, see *figure 1*.

Figure 1: Author's TMA editing dialogue box.

This enables the author to setup a TMA template which includes introductory information, warnings to students, the JavaScript snippet, and submission information at the end. Authors are prompted for total marks, numbers of questions, and numbers of parts, all of which are set up for them in the template in the format agreed upon by the module team. (Main) questions and question parts are dealt with slightly differently, but for both authors are asked whether they want to setup the question/part as a stem or to actually ask for a student response. A stem is introductory information, such as a scenario, which the following question parts or sub-parts will actually ask about.

When setting up a question part, the author is prompted for all required information which enables the plugin to put in the correct headings with question part details, marks and so on. All of the required areas are produced, with a question area and instructions for extending to include additional code or text boxes if needed, an answer area with pre-written prompt in both text and code formats (the un-needed one can be deleted), a solution box, bullet-pointed marking breakdown and all agreed prompts.

It may seem as if there are options missing from the dialogue; the buttons provided are all that the authoring team has needed to date although should additional functionality be needed it is available and just needs a button creating for it.

3.3.2 Marking Plugin

An additional plugin was developed for the tutors to support their marking. This added a number of buttons to the toolbar, as shown in *figure 2*.

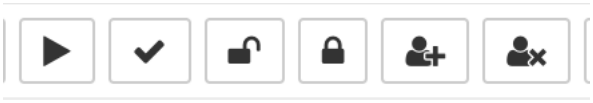

Figure 2: Marking buttons

The first tool prepares the TMA for marking. This is analogous to the preparation script from the previous incarnation. This is done none-destructively, with the Jupyter Notebook being duplicated and the new copy having -MARKED appended to the filename. A feedback box is added to each student answer, along with a separate box which prompts the tutor to enter the marks for that question part (with the amount available also shown). Colour coding is automatically applied.

One of the problems with the earlier iteration was the student's answer cells being un-editable. The lock/unlock buttons fix this by removing and reapplying the editable metadata on all of the student's cells in one go, meaning a tutor can comment on or run student answers and then relock them so as to not inadvertently edit them. This also enables additional feedback and mark boxes to be copied in if a student did happen to do something odd and cause them not to be generated correctly.

The person buttons were in response to a tutor who complained that, when marking question by question in this way, they struggled to remember whose assessment they were marking and hence struggled to make the marking personal. Personalised feedback has an increasingly effective impact on improving learning (Perez-Segura, Ruiz, Gonzalez-Calero, & Cozar-Gutierrez, 2022). The person with a plus button adds some code to temporarily make the student's name cell (the first cell in the Jupyter Notebook) float at the top of the screen regardless of scrolling. The person with an x button returns the box to its rightful place at the top of the script.

The tick button iterates through all of the mark boxes and creates a table of marks at the bottom of the assessment. Any marks that are missing are highlighted and the tutor warned. Very occasionally, students inadvertently duplicate answer fields, which can cause duplicate feedback and marking boxes. This is also highlighted to the tutor. The table of marks correlates with the assessment recording system the University runs so the marks can be transferred over easily and the totals compared, reducing the chance of human errors.

3.4 Additional Tools

Further tools have been developed to support tutors and students which fall outside of the realm of Jupyter Notebooks but which nevertheless support assessment writing, marking and completion. Perhaps the most useful of these is *allowed* (Wermelinger, 2023), which is designed to enable assessment authors, tutors, and most importantly students to ensure their code does not deviate from the agreed and taught subset of Python used within the module. The module has strict guidance that any students using constructs not taught within the module receive zero marks for the question. This was originally designed to restrict users from using, for example, list comprehension which made assessing algorithm complexity (a central tenet of the module) more difficult. However, there are always get some students who, deliberately or inadvertently, submit work which breaches this rule. The *allowed* tool reduces the chances of these mistakes occurring by allowing students to check their own code in advance. This in turn, reduces the unnecessary stress on students while making the assessment process more accurate and simpler for the tutor since they can also run the tool to detect problems.

4. Conclusions

Thousands of TMAs have now been completed in Jupyter Notebook, submitted and marked within M269. While not all students enjoy having all module material within a single location, with some wishing to have a print book, there have been no complaints regarding the assessment material being self-contained within the Jupyter Notebook.

There have been no occurrences of tutors and students ending up with differing version of the assessments, and in-presentation changes have been much simpler in the second presentation using the plugin than in the first presentation using the Python scripts.

In the second presentation the module team have been more active in monitoring tutors' marking. There have been minimal complaints from tutors with regards to the uptake of the marking plugin in comparison to the scripts used in the first presentation and although one tutor has experimented with an alternative marking methodology, all other tutors have marked using the plugin, with no issues reported to or identified by the module team monitors.

There is no comparative data available as to the level of cognitive load experienced by students either before or since the change to Jupyter Notebooks. However, casual conversations with both tutors and students have suggested that the methodology works and cognitive load in terms of document switching and multi-file management is minimised.

Further exploration is needed to quantify improvements as well as to identify and develop further functionality within the authoring and assessing plugins, as well as additional tools to support tutors, module team and students through their learning journey. A group of like-minded tutors and students both past and present has been set up to explore these opportunities.

5. Acknowledgements

While the assessment process, tools and plugins were written by the author, their development and testing was a team effort; I thank Michel Wermelinger (production chair), Brendan Quinn, Adam Linson, Phil Hackett and Jane Evans for forming a well-knit and forward thinking module team and all of the M269 tutor team for being willing to trial and feedback upon new technologies.

6. References

Al-Gahmi, A., Zhang, Y., & Valle, H. (2022). Jupyter in the Classroom: An Experience Report. *SIGCSE 2022: Proceedings of the 53rd ACM Techincal Symposium on Computer Science Education* (pp. 425-431). Providence, RI: ACM.

Cavanaugh, A. J., & Song, L. (2014). Audio Feedback versus Written Feedback: Instructors' and Students' Perspectives. *MERLOT Journal of Online Learning and Teaching, 10*(1), 122-138.

Cerdan, R., Candel, C., & Leppink, J. (2018). Cognitive Load and Learning in the Study of Multiple Documents. *Frontiers in Education, 3*(59), 1-7.

Davies, D. J., McLean, P. F., Kemp, P. R., Liddle, A. D., Morrell, M. J., Halse, O., . . . Sam, A. H. (2022). Assessment of factual recall and higher-order cognitive domains in an open-book medical school examination. *Advances in Health Sciences Education, 27*, 147-165.

Hahn, M. G., Navarro, S. M., Burgos, D., & De Lan Fuente Valentin, L. (2021). A Systematic Review of the Effects of Automatic Scoring and Automatic Feedback in Educational Settings. *IEEE Access, 9*, 108190-108198.

Hirst, T. (2022, 4 6). *Notes on the JupyterLab Notebook HTML DOM Model, Part 1: Rendered Markdown Cells* . Retrieved from OUseful.into, the blog...: https://blog.ouseful.info/2022/04/06/trying-to-make-sense-of-the-jupyterlab-notebook-html-dom-model-part-1-rendered-markdown-cells/

Jupyter Notebook Extensions. (2023, 07 19). Retrieved from GitHub: https://github.com/ipython-contrib/jupyter_contrib_nbextensions

Lafleur, A., Côté, L., & Leppink, J. (2015). Influences of OSCE design on students' diagnostic reasoning. *Medical Education, 49*(2), 203-214.

Nutbrown, S., Higgins, C., & Beesley, S. (2016). Measuring the impact of high quality instant feedback on learning. *Practitioner Research in Higher Education, 10*(1), 130-139.

Perez , F., & Granger, B. E. (2021). Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science & Engineering, 23*(2), 7-14.

Perez-Segura, J. J., Ruiz, R. S., Gonzalez-Calero, J. A., & Cozar-Gutierrez, R. (2022). The effect of personallized feedback on listening and reading skills in the learning of EFL. *Computer Assisted Language Learning, 35*(3), 469-491.

Suoto, I., & Nadas, R. (2007). The 'Marking Expertise' projects: Empirical investigations of some popular assumptions. *Research Matters, 4*, 2-7.

The Open University. (2023, 07 03). *M269*. Retrieved from The Open University: https://www.open.ac.uk/courses/modules/m269

Wermelinger, M. (2023). Checking Conformance to a Subset of the Python Language. *2023 Conference on Innovation and Technology in Computer Science Education.* Turku, Finland: ACM.

Willis, A., Charlton, P., & Hirst, T. (2020). Developing Students' Written Communication Skills with Jupyter Notebooks. *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)* (pp. 1089–1095). Portland, Oregon: ACM.