Pre-course suggestions

University of Roehampton

Prof Miles Berry

2025-2026

I'm delighted that you've chosen to train at Roehampton as a computing teacher, and am looking forward to working with you over the year ahead: both on campus and in school.
You've a couple of months over the summer to prepare for the course. I've made some suggestions below, but please don't see these as compulsory or limiting. Your future work as a computing teacher will draw on three bodies of knowledge: your subject knowledge in computing, your skills as a teacher and your ability to connect these so that the pupils in your care develop their own capabilities in computing. The suggestions below address all three of these areas.

The DfE's initial teacher training and early career framework details the minimum expectation for the content of training courses, and also will form the basis for your induction period as an Early Career Teacher after you achieve QTS. Look over the 'know that' statements, highlighting any that you are surprised by, disagree with, or would like to discuss. The bibliography is *excellent*: I'd recommend starting with some of the readings for 'How pupils learn' and 'classroom practice'.

You'll also become very familiar with the Teachers' Standards, which still form the basis for the award of qualified teacher status (QTS). At this stage, read over these, highlighting any that you currently have concerns about.

There are some excellent, accessible materials from the Chartered College of Teaching, aimed at those joining the profession, in their Early Career Hub - the College is *free* to join as a student.

The Education Endowment Foundation have reviewed the evidence base for intervention strategies in schools - not all of these are relevant to computing education, but their Teaching and Learning Toolkit provides a good route in to some of the relevant research. You should consider how secondary schools might make use of these approaches in teaching computing. If you're interested in engaging with the academic literature directly, try Muijs et al (2014)'s wide ranging review as a starting place.

Rosenshine's Principles of Instruction have been adopted by many schools as a

basis for effective lesson design - whilst these are open to critique, you should consider the relevance of these to your teaching of computing. Here's Rosenshine's list:

- Begin a lesson with a short review of previous learning.
- Present new material in small steps with student practice after each step.
- Ask a large number of questions and check the responses of all students.
- Provide models.
- Guide student practice.
- Check for student understanding.
- Obtain a high success rate.
- Provide scaffolds for difficult tasks.
- Require and monitor independent practice.
- Engage students in weekly and monthly review.

Much of our current thinking about effective teaching and learning is based on psychology and cognitive science. I think Didau and Rose's (2016) What Every Teacher Needs to Know about Pyschology is a good general introduction.

To keep up to date with what's happening, read the TES, Schools Week and the education coverage in the Guardian.

Your subject knowledge for computer science, information technology and digital literacy is crucial for your impact as a teacher of secondary computing and this summer provides an excellent opportunity to develop this as far as you can. Programming is the most significant element of the national curriculum and the GCSE and A Level exam specifications, and this should be your focus.

Python has, for good or ill, become the language of choice for most secondary computing, so developing your fluency in Python coding is probably the most useful thing to do. There are some excellent online resources for doing so. Depending on your own level of expertise, you might find one or more of these helpful:

- Runestone Interactive: How to think like a computer scientist
- Runestone Interactive: Problem solving with algorithms and data structures using Python
- Datacamp: LearnPython
- Python for you and me

Rather than working through excellent resources such as these, you may find it more engaging to have a go at actually programming something, and at the least

you should be able to compare and contrast your experience of learning though working through tutorials with that of solving problems or working on projects.

A great source for programming challenges or problems is Project Euler. A few of these are mirrored in the rather good set of programming challenges made available by OCR. You should have a go at some of these, from either set.

If you're interested in object oriented programming (and if you're going to teach at A Level, you should be), Python's game library, Pygame, is a good, motivating place to start, perhaps using the PythonProgramming tutorial.

An alternative is Processing, which is Java based, although there's a Python syntax version available, but the most interesting work at the moment is done using the online JavaScript version, p5.js. Daniel Shiffman has produced excellent resources for Proecessing and p5.js, including some very engaging video tutorials for his Coding Train channel on YouTube, and an in depth exploration of creating a physics engine in The Nature of Code. For p5.js, New York City CS4All have an excellent computational media course for high schoolers.

If you've never explored a block-based (rather than text-based) programming language, you really should do so. MIT's Scratch is hugely popular in primary computing education, and is often used for introductory work in Year 7, and sometimes Year 8. One of the best aspects of Scratch is that young programmers routinely share their work with one another through Scratch's online community and are encouraged to read and remix others' code in their own projects. Have a look at what young programmers are capable of outside the formality of school, perhaps leaving constructive comments on some of the projects you explore. UCBerkely's Snap! deserves to be better known and used: it extends Scratch's block based programming by adding in the implementation of some more sophisticated programming concepts. Modrow's (2022) Computer Science with Snap! covers a lot more CS than we would in secondary school, and begins with a relevant example of the spread of an epidemic...

More generally, there's excellent support for developing your subject knowledge for the GCSE and A Level specifications from the National Centre for Computing Education ('Teach Computing'), through their Isaac Computer Science materials and their courses. There are a couple of *face to face* courses running over the summer, which might be of interest.

The wide applicability of 'computational thinking' has been used to justify the inclusion of computer science and programming in the curriculum. Whilst there are many ways in which programming can be applied to solve problems in and beyond other academic disciplines, there is rather less evidence that the **ideas** and **approaches** of computer science transfer comfortably to other domains. For now I'd recommend reading Tedre and Denning (2016) The long quest for computational thinking, which is the set pre-course reading, and perhaps Jeanette Wing's (2006) CACM article 'Computational Thinking', which is widely credited with coining the term.

The national curriculum programmes of study for computing are admirably brief, but we expect changes soon following the recent curriculum and assessment review. They set out the scope of what pupils should be taught, but leave the details of implementation and assessment to individual schools or teachers. You should be aware that at present it's only local authority schools that are legally required to follow the national curriculum. It's worth mentioning here that the national curriculum subject, computing, encompasses computer science (foundations), information technology (applications) and digital literacy (implications) - there's more to computing than computer science, and there's more to computer science than coding. I wrote a guide to the Key Stage 3 programme of study for the BCS/CAS, which may be of interest. The Raspberry Pi Foundation Big Book of Computing Content is an excellent collection of accessible magazine articles.

Whilst computing is a national curriculum subject at Key Stage 4, evidence suggests its rarely treated as such. An increasing minority of pupils are entered for the GCSE in computer science, with around 90% of these following OCR's specification. There is some controversy over the place of practical programming here: from 2022 entries onwards, programming has been examined by OCR through a written exam, with schools certifying that pupils will have had the opportunity to engage in unspecified practical programming as part of the course. Government have accepted the recommendation that GCSE computer science be replaced by a GCSE in computing generally, and the requirements for this new qualification are in development.

There are relatively few entries for computer science A Level, although the exam specifications here are engaging and rigorous. In my view, AQA has the edge, in part because it includes some functional programming and data science. There's much scope here for ambitious and well-prepared students to develop outstanding, independent project work that would stand them in very good stead for admission to university CS degrees. Even before the development of the computing programmes of study began, the DfE commissioned an expert group to draw up recommendations for subject knowledge requirements for those seeking to train as computer science teachers. These formed the basis of the audit you undertook prior to your interview, and you'd be wise to spend time addressing any areas where you still feel your expertise may be lacking.

Much of the taught component of your PGCE will focus on approaches to planning, teaching and assessing computing (or your intent, implementation and impact as Ofsted now characterise these). For now, let me point you in the direction of some research-informed frameworks for effective CS teaching:

- Paul Curzon on learning to learn to program
- Neil Brown and Greg Wilson (2018) Ten quick tips for teaching programming.
- Colleen Lewis CSTeachingTips website.

- How we teach computing from Teach Computing
- Michael Kölling and John Rosenberg (2001) https://kar.kent.ac.uk/13607/1/guidelines_for_teaching_objec

What common ground do you notice? Are there significant differences between these lists? What would explain that?

If you'd like to read more about the teaching of programming (and other aspects of computing), I'd recommend these five books as a starting point:

- Guzdial, M., 2015. *Learner-centered design of computing education: Research on computing for everyone*. Synthesis Lectures on Human-Centered Informatics.
- Lau, W., 2017. *Teaching Computing in Secondary Schools: A Practical Handbook*. Routledge.
- Sentance, S., Barendsen, E. and Schulte, C. eds., 2018. *Computer Science Education: Perspectives on Teaching and learning in school*. Bloomsbury Publishing.
- Harrison, A., 2021. *How to teach computer science*. John Catt.
- Coleman, G. (2021) *The big book of computing pedagogy*. Cambridge, UK: Raspberry Pi Foundation.

For the academic component of your PGCE you'll need to write three essays. We'll cover the requirements for these in detail as part of the course, but you might like to spend some time over the summer on developing your writing. I'd advise you to avoid an overly 'academic' style, aiming instead for clear, precise prose. Success in 'academic' writing is as much about the reading and the thinking that comes before the writing - the more you read, and the more you think about what you read, the better your writing will be. Good academic writing has much in common with good programming:

- You should start with a clear understanding of what you want to achieve.
- You should have a plan, both overall and at the detail level of sections and paragraphs.
- Manage complexity: focus on what matters, and recognise the levels of detail encompassed by sections, paragraphs and sentences.
- Syntax matters, but semantics matters more.
- Test frequently - check that your argument makes sense, is logical and supported by evidence.
- Explain what you're doing.
- You must acknowledges the source of ideas and text.
- As the Zen of Python has it, **'Simple is better than complex. Complex is better than complicated.'** I'm sure there are other parallels, and I

now appear to have contradicted what I wrote above about limited evidence for the transfer of computational thinking. When you're here, you'll have access to good support from the University's academic achievement team. They recommend Gillett, Hammond and Martala's

(2009) Inside Track to Successful Academic Writing.

Let me conclude with some suggestions for particular technologies which you should be able to draw on in your study and your work as a teacher.

- Screen recording - we've come to recognise how useful it is to record a presentation, software demonstrations or a programming example. Windows and MacOS have built in screen recorders, and their are online tools available, such as Screencastify.
- Note taking software. Microsoft OneNote is very popular in schools, and can be used as a class. Others are happy with well organised documents in Google Apps.
- Quizzes. It's really useful to be able to create interactive, self-marking quizzes for pupils, for assessment and to help them to remember what they've been taught. I like Google Forms running in quiz mode, but Kahoot is very popular in schools. You might also explore Project Quantum's computing questions available in Diagnostic Questions.
- PDF annotation. You'll find yourself reading lots of PDFs - annotating these online is much better than printing them off! Mendeley is good, and also helps with bibliography management.
- Online IDE. It's useful to be able to write and run code online, without having to install anything locally. Raspberry Pi's editor has a good set of features. https://trinket.io/ is a little simpler. I've a personal preference for Jupyter Notebooks, using Google's Colab system. For those of you who would prefer to go with an offline environment, you can't do better than VSCode.
- Online office software. You'll also need to be able to work collaboratively on documents and presentations - Google Apps and Microsoft 365 are both very good: the University provides Microsoft 365.
- Email and Calendar. The University's Outlook provision should suffice for this - install the mobile app too, and use it to manage your timetable, lectures, assessment deadlines and social life.

I'm taking some leave over the summer, including some time beyond the reach of the net, but please do get in touch if you'd like to discuss any of the above, or have any other questions or concerns about starting the course.

Prof Miles Berry
m.berry@roehampton.ac.uk