My 10 lessons for the CER field

Preface
This was written for my 5 min. talk at the CCSE Festival 29 March 2022.

Table of Contents -- List of the "ten"—> 0 to 11 lessons.

Related rough Memos I have written recently, and might draw on in addition to the 10 lessons
"ExpertiseNote.docx" Feb-March 2022. The argument(s) about expertise as an objective in some learning.

"JParkRemediation3.docx" Dec 2021 and on to April 2022 and beyond. SpSkills,

"LeoPorterPLangLearning.docx" Jan 2022

"InformalLearning.docx" about informal learning; Nicola Looker's work. But perhaps it is good to look at non-HEI learning -- areas where we do learn but not from teachers or to an explicit syllabus.

"QuintinNewBigIdea.docx" in folder "LTPfrom2021".

"JSingerPaper.docx"

Other memos which have been almost immediately triggered by writing/talking about these ten lessons

"QuintinNewBigIdea.docx" -- major extensions to this TM by me, applying the lessons to L&T of a first PLang.

Lesson 0 General education (or "LTP" - learning and teaching process) theories, not discipline-specific ones.
Putting general theories of education first and the content-matter-discipline second leads to different perspectives than the other way round. Since I came to GU and attended the then Centre for Science Education, been interested in general education theories, not discipline-specific ones; but most Ts including most of CER is based on a single discipline.

This memo is about applying general educational principles or theories, not discipline-specific ones.
The "lessons" that follow are not randomly but partially ordered. Putting general theories of education first and the content-matter-discipline second leads to different perspectives than the other way round. N.B. Lee Shulman (2005) made content-centered perspectives fashionable again with his notion of "disciplinary pedagogies".

- https://www.psy.gla.ac.uk/~steve/localed/lcomm.html - sigpeds
- https://files.eric.ed.gov/fulltext/EJ697350.pdf Liberal Education paper
- https://www.jstor.org/stable/20027998 Daedalus paper

⇨

Most HE teachers know enormously more about what they wish students knew, than about what the students actually do know when they start. All about the destination, very little about the departure point. **See also lesson 7**, which is essentially the same point. But here it is a foundational point suggesting that a lot of problems are basically from ignoring constructivism; and from the longstanding and persisting tradition of hiring content matter experts to teach: in CS/progging, in all HE, in traditional craft guilds.
This is also the LTP dimension of gradient of expertise between teacher and learner.

Lesson 1 Self-paced vs. synchronised learning
Self-paced learning (e.g. solo learning from a book) has an enormous advantage over learning in class. The learner can go at their own pace, and every learner will go slow and fast at different points.
(Solo video with a pause/rewind control is self-paced.)

Synchronised learning is inconvenient to arrange AND then inefficient for each learner. But has some essential virtues – typically, dialogue.
N.B. Chi has shown that pairs watching a video together, with a worksheet can be as good as having a live tutorial. The worksheet keeps them on task; but the video controls allow taking time when needed.

Synchronised learning is inconvenient to arrange AND then inefficient for each learner. But has some essential virtues – typically, dialogue.

N.B. Chi has shown that pairs watching a video together, with a worksheet can be as good as having a live tutorial. The worksheet keeps them on task; but the video controls allow taking time when needed.

Lesson 2 Not only conceptual knowledge but **expertise** is required for even Introductory programming.
Much learning is so that given plenty of time, you can retrieve and apply things you know. But to write a program, or follow a lecture that puts up code and discusses it, you cannot work if you are looking up stuff. It has to be instantly recalled (within a second or two) or the learner cannot follow.
Maths up to (say) end of level 1 or 2 is like this; and learning a PLang is like this.

This means expertise is the standard of learning / performance: a much higher standard than for most academic learning.
It is measured in response time to a question.
You get expertise by a LOT of practice (most often self-paced), each time on a new "problem", even if each is only a small variation on the one before.

Lesson 3. Exams tell the researcher nothing about how good or bad the learning or the teaching is: only about how much the learner knows at the end.
Exams are a service for employers; but largely valueless for teachers. Unless you know how much students knew to start with, you do not know how much they have learned.
You need to have a test which is used twice: PT1 is the pre-test, PT2 the post-test.
And the test needs to evoke the same learner motivation, and so not to have marks, grades, credit awarded for it.

The **gain** is PT2 - PT1. (How much the course added to the learner's knowledge.)
100% - PT1 is **the number of questions or marks got wrong**, and so what could ideally be learned during the course.
The **normalised gain** is (PT2 - PT1) / (100% - PT1) i.e. if the course was totally effective.
This allows courses (and institutions) to be compared despite having very different kinds of students when they arrive.
See Hake (1998) doi: 10.1119/1.18809
Thus studying your own course often has a big problem unless you address this, because you have no measure of the amount of learning; only measures of how much expectations were satisfied.

Lesson 4 Study not only mean performance / effect in the class as a whole, but also **study individual variation**.
It is not unusual to find several groups of students within a class who "perform" or otherwise behave rather differently from other groups. Consider routinely investigating this, and preparing reports on how each group performs (or conversely: how well or poorly the course as it is serves each group). The most obvious groupings are gender and race, but type of motivation (for marks vs. for intrinsic interest) are another common grouping; and usually neglected: how much each group knew before the course started. If students lack prior knowledge compared to other groups, they are likely to fall increasingly behind if the teacher assumes this knowledge i.e. the course is designed only for a subset of students.
Lesson 5 There are **two quite different learning motivations** in play in programming.
Desire for marks, and Intrinsic interest in doing this stuff (which means: there is nothing such a student would rather do, so they spend all their available time on it). Courses tend to be adapted to the students they have by assuming knowledge picked up in hobbies, and assuming that 16 hours a day is what their students do. This might be a good design for an elite course, but not for a course supposedly suitable for all students.
See my Margolis web page for an introduction to the literature on the major change at CMU to their undergraduate course, where a study by Margolis showed how very different the male undergraduates were before the change compared to highly competent and hard working, but not fanatical, female students; and how the course was adapted / tuned for the males.

Lesson 6 Time On Task (ToT) is often the biggest factor (intermediate measure and causal factor) in learning.

The time that matters is time processing new thoughts and examples, not sitting passively or "actively" pressing buttons.

If you don't believe this, then find real quantitative evidence. Inspiring students, or amusing them, makes them spend more time .... Good idea when you can get it to work.

But also habits, and practice required and done (as in sports training) makes a big difference in ToT, and so in learning accomplished. [Fionnuala produced a counterexample to the bald statement of how over training reduces performance in sports people.]

Lesson 7 The natural blindness of subject matter experts used as Ts e.g. CS faculty.

The natural blindness of Ts who are subject matter experts e.g. CS faculty: you teach as you were taught yourself; *and* you assume your students already know what you knew then.

This is natural in all disciplines, but may be a bigger effect in CS because it is a young subject:
Cambridge histories:
1953 one year course at u/g level.
1971 One-year Computer Sciences Tripos, first independent undergraduate teaching

Maths has a much longer history than CS (400 years since Tudor schools for non-monks i.e. clerks for merchants and Kings began, > 2,000 years since empires with tax records began and so presumably servants being taught maths) i.e. maths has had much longer to improve on the teaching methods it began with.

Lesson 8 Hidden ILOs

An ILO is an Intended Learning Objective. Some are explicit; but there are many more which are not explicit, but which are as important as the overt, explicit ones. They may be deliberately hidden, they may be implicit: unstated but picked up by Ls. Some may be unknown / unstated to most of the teaching staff, and only known and maintained by a few senior staff.

What would be involved in taking the time to articulate for (and consciously plan from) each course and programme what its ILOs really are -- normally they are only what is examined, and omit longer range goals, social goals, learning to learn, etc.etc. I have written about this in one or two published papers with JoeM.

Would we do better to teach these more directly? or not? Would we (or not) do better by stating these explicitly? Generally I think it is good for teachers and course designers to be explicit with themselves about all of them, so as to design better, check they are being achieved etc.

Sometimes it is unimportant to learners to be told about them from the start, but it may be useful to eventually point them out -- perhaps as a reflective lecture at the end of each year. Cf. Dzillias work on programmers only understanding years into their post-HE careers, just how useful their CS education was.

Some types of hidden ILOs

a) Learning how to learn in the discipline is a huge part of what a student / undergraduate learns. And though the methods may clump amongst some other disciplines, yet there are big differences in how to learn between some other disciplines. This is learning how to teach yourself (more material in the same discipline).

b) Learning to judge work by yourself and others: internal quality control. Even writing a sentence in English calls for a lot of this; and in Maths and Computer Programming, you learn to look at a formula or expression and check it in many ways instantly. This "evaluative judgement" is essential in all subjects.

c) Sfard suggests that another essential but largely unrecognised part of learning is learning to participate in the social relationships around the knowledge (being a doctor vs. just knowing a lot of facts and procedures; being a software engineer as opposed to just writing code).

d) Social ILOs -- achieving good social relationships amongst the learners (e.g. for useful discussion and collaborative learning). Mentioned in the "Thinkathon" paper: doi: 10.1145/3304221.3319785

e) Practice at related skills (and graduate attributes, professional skills) including: giving a talk, writing an essay, writing a computer program, working in a team to produce a joint product, ...

f) Must teach a fashionable PLang. If as a teacher this is an objective, then you should write out for yourself, the pros and cons of this. E.g. if fashionable it may be popular in job ads. But what is good for immediate commercial use is NOT necessarily or even probably good for learning the basic concepts. (I always remember as part of my physics degree doing a few hours on basic workshop skills e.g. turning brass on a lathe. The senior technician in charge of this mentioned that he reserved the best lathes for this: it takes more skill to operate a worn and faulty machine than a fresh one with no worn and faulty parts.)

g) Must teach an enjoyable (for the learners) PLang. Similarly: if this is a requirement for the PLang you choose to teach, you should be able to list the pros and cons of this. Good for intrinsic motivation but bad for serious students motivated extrinsically by future job opportunities. So if you think motivation is your reason, you have to know that in your particular class, most students are there for immediate pleasure but not for the value to their career.

h) In learning a new PLang (Programming Language), why do some of us not bother memorising lexemes ("else if" vs. "elsf"), nor syntax but just look them up every time? We just learn plans (which are often cross-PLang). If this is so, then should teaching introductory programming be about teaching plans and *not* vocabulary and syntax. Or is it taught like that? (in which case this paragraph may contribute to explicit theory, but not to practice).

Lesson 9 Basic constructivism and its implications

A] The start and end points are both important in learning; and so should be (must be) important in teaching.

B] Telling doesn't work well <u>because</u> no processing, no linking of new knowledge to other knowledge in the learner's mind.

See my NATO paper written in 1994, where I discuss the relationships between a number of educational theories and ideas, including constructivism.

Lesson 9b Basic constructivism and its implications (continued): types of learners' prior knowledge, which must be identified and planned for by the T.

This includes, ideally, identifying the multiple types of prior knowledge of Ls:

1. Concrete examples in the world Things like process thinking: that could be used as concrete links from new knowledge to experience in the world. Physics has done this well; CS not.
2. Spontaneous conceptions: (a type of misconception). "Traps" that many Ls are prone too where something which conceptually is not really connected to new material, is widely seen as connected e.g. food for plants taken up in their roots.
3. Required pre-requisites for a course which are not acknowledged but assumed cf. Margolis; spatial skills.
   These may be (i) bits of conceptual knowledge; but they may equally be (ii) familiarity with the problem scenarios that it turns out a course depends on or uses extensively e.g. numerical and algebraic tasks.

Lesson 10 Why many students found covid isolation very destructive of their university experience.
Most students use their peer group to help their self-regulation of work. Self-regulation means something rather like scheduling in an operating system (OS): deciding what tasks to do next, and when to do them, and making these decisions under dynamic conditions. I.e. to-do lists and careful plans are not adequate to be a student, where every task requires an unknown (to the student) amount of time and effort exactly because learning largely consists of being stretched to do each new task. The teacher probably hasn't even tried the task themselves, and even if they have this is no more relevant an estimate than a toddler's speed over a course is for a trained athlete's speed.

Even more basic is that pre-covid, students would check with their classmates what was and wasn't meant by any instructions they received. They could generally tell by looking at the facial expressions around them; and equally could ask students who did attend a lecture what was expected of them. During covid most "classes" were like getting a DIY kit with written instructions, versus being able to talk to an expert as you try to make sense of them: not just, no sense of human social contact, but very poor communication as measured by the certainty of the recipient when receiving instructions for action.

Lesson 11 The effects of expectation on learning tasks: How teachers often reduce learner performance inadvertently. (This lesson perhaps should really come directly after (5) on intrinsic and extrinsic learner motivation. It is a general point in psychology that performance is influenced apart from what is usually called "motivation" -- more about influencing run-time behaviour than stating goals.)
All tasks we do are generally under-specified; and the pervasive tradeoff is between speed of execution vs. error rate. There are cases where specifying an expected speed halved the speed it was done at. This may apply to learning tasks. Sugata Mitra's work (mostly in the form of online videos) could be interpreted in this way. He has repeatedly demonstrated how much better children often do at a learning task if teachers are prevented from setting goals. Children still in fact acquire learning goals from the wider culture or sometimes from Mitra himself.
This may also explain what Papert was getting at in his studies of children exploring what can be done with computers. In contrast, computer science teachers generally have the attitude that the SoftEng industry is about eliciting a "spec" and implementing exactly what the customer requires and NOT exploring what can be built creatively; and the teachers replicate this attitude. But Papert and some others have argued that this is wrong. This then is another dimension of how PLangs can be taught. And perhaps it is a dimension of the difference between how a person teaches themselves a new PLang vs. what a lecturer will demonstrate and say in teaching students a PLang.
My notes on, and pointers to, Sugata Mitra's work

Great papers touched on here

Hake,R.R. (1998a) "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses" *Am.J.Physics* vol.66 no.1 pp.64-74 doi: 10.1119/1.18809 PDF copy

Sfard,A. (1998) "On Two Metaphors for Learning and the Dangers of Choosing Just One" *Educational Researcher* Vol.27 No.2 pp.4-13 doi: 10.3102/0013189X027002004

M.A.K. Halliday (1985) *An introduction to Functional Grammar* (Edward Arnold: London)

Histories of teaching computing at Cambridge University:

(1) History of the University of Cambridge Computer Laboratory

About six pages compiled by Karen Sparck Jones. Version 6.0, 20 December 2001.
Mentions what kind of CS courses were started when:

- 1953 One year diploma course at u/g level.
- 1971 One-year Computer Sciences Tripos, first independent undergraduate teaching
- 1985 One-year MPhil in Computer Speech and Language Processing
- 1989 Full 3-year Computer Science Tripos began
- (1990 92 PhD students, 33 Diploma, 170 Tripos, 19 MPhil)

(2) History of the University of Cambridge department of Computer Science and Technology

Department of Computer Science and Technology web page, with a brief history of it leading to more details, including: ...

(3) Haroon Ahmed has written an extensively illustrated, highly readable and informative account of computing in Cambridge, from Babbage to the present day. His book is available as a PDF download entitled Cambridge Computing – The First 75 Years .

https://www.cl.cam.ac.uk/events/EDSAC99/history.html
https://www.cst.cam.ac.uk/history
http://www.cl.cam.ac.uk/downloads/books/CambridgeComputing_Ahmed.pdf