Teaching a Lab

Before the semester even begins
There are a number of things you will need to know to ensure your lab goes smoothly and successfully: administrative details for the course, your and other people's responsibilities related to the lab, expectations and policies, grading criteria, and safety measures. You can find a checklist at Things To Do Before Classes Begin: *Checklists, from CRLT*.

You should also have handy contact information for other course instructors, lab technicians, and emergency service personnel. An example Laboratory Course Contacts List, *from CRLT*.

Finally, it is important to decide on the main objectives for each lab as well as the criteria for assessment before the start of class. Tips for how to set up assessment criteria are found at: Best Practices for Grading Lab Reports, from CRLT, and example rubrics for grading can be found at: Sample Laboratory Report Rubrics, from CRLT.

Preparation and planning during the semester:

- ❏ Ahead of the lab, **perform the lab assignment** (give the full lab time for this), thinking about it from your student's perspective. **Anticipate** and plan for what students may have trouble with, or anything that may go wrong.
- ❏ Determine and review what **prior knowledge** and skills students need to succeed in the lab. Are there any common misconceptions or issues? Do they need a demonstration of equipment or software features to get them all started on the same page?
- ❏ Determine **how the lab connects** to lecture, to the big picture, and/or the real world.
- ❏ Determine the **learning objectives** (the main 2-4 things students are going to learn or practice in the lab).
- ❏ Write down **potential questions** that you can ask your students to gauge whether they are ready for the lab, or are understanding the lab.
- ❏ Think about **time management** (how long to allow for each task).
- ❏ Determine any **security concerns** and decide how you will make students aware of them.

When you are finally in the classroom, you should plan to spend your time walking around to the various individuals or groups of students. Telling students they can come ask you questions rarely works as well as setting the expectation that you will be circling, stopping at each group, and asking them questions to help gauge their understanding and progress. Getting students talking, especially in groups, is not always easy. Tips for developing strong questioning skills are available at: Strategies for Managing Discussions With Groups in the Laboratory Class.

After students finish an experimental or computational lab, they are usually graded on their performance. Assessments can range from checking that a program executed correctly to grading a written report. Tips for grading lab reports, including how to set up criteria, can be found at "Finally, it is important to decide on the main objectives" section.

2.2b - Computational Labs

During Labs

- ❏ **Help Policy:** Determine your policy for answering debugging questions. One possible policy is to require students to complete/attempt General Debugging Steps 1-3 listed below, BEFORE they ask for help. This encourages students to debug before asking for help.
- ❏ **Queue Policy:** Determine your policy for the order in which you answer questions. One possible policy is to assign numbers to students who have questions or ask them to write their name in a queue on the white board. This lets students continue to debug/work while you are helping others as opposed to students standing in line.
- ❏ **Collaboration:** Determine if students are allowed to collaborate during lab and help each other debug. If collaboration is allowed, encourage students to help each other.

TIP: Encourage students to think about why and when the problem occurs. Have them explain to you on paper what happens to the input and the variables as they walk through the program. Ask them if the program actually does what they say it does. If not, when does it veer off. Help guide students through debugging.

General Debugging Steps (adapted from Advanced R by Hadley Wickham)

1. **Reproduce it:** Make sure you can reproduce the error before you start debugging.
2. **Reduce input:** Determine the smallest input that causes the error. The smaller the input, the easier it will be to find the error.
3. **Isolate problem code:** Isolate the portion of your code causing the error. You can do this by tracing the data's flow through your program. At the start of each function, do the variables contain the values you expect? Do the functions return what you expect? Isolating the function or lines of code causing the error will help you find the solution.
4. **Experiment:** Hypothesize a potential cause for the bug. Then test to see if your hypothesis is correct by changing the input or code to either rule out the hypothesis or confirm it.
5. **Experience:** Think if you have had this type of error before and what the solution is. Do an online search or talk to others. Sometimes, explaining the problem will help you discover the problem.
6. **Never Give Up:** This does not mean you cannot ask for help.

More resources:

- IT safety policies
- EECS Tutoring Resources and Engineering Center for Academic Success
- Brown NCC, Wilson G (2018) Ten quick tips for teaching programming. PLoS Comput Biol 14(4): e1006023.
- Lewis C. CS Teaching Tips; 2017.