CS Tutoring Tips

From "Tips for CS Tutors and Mentors",
by Weinman and Rebelsky, Grinnell College

September 4, 2015

Thank you for becoming a CS grutor! The CS department would not be the same without our army of dedicated grutors. The help you provide students not only improves their CS skills, but also helps them get to know the department and informs their opinion of the program.

Expectations of CS Tutors

Stay up to date on the course assignments. Before your tutoring session, spend at least five minutes familiarizing yourself with the current assignment.

Your primary goal during your hours is to support your peers. Make sure you introduce yourself and check in with students at least every 10 minutes even if no one is asking for help. Someone might need help but might not want to interrupt you.

Your secondary goal during your hours is to work on other tasks related to the course. Talk to you faculty member about what other tasks are available. These tasks might include:

- Grading assignments
- Answering questions on Piazza
- Looking at upcoming homework assignments or grading standards to offer suggestions
- Developing review questions for content in the course that students could use
- Reviewing course materials to improve your understanding of the course content

The goal of ALL faculty members is for you to always have something to work on even if there are no students to help. Please let your faculty member know if there are no remaining tasks to work on. Only after having sent an email request to your faculty member would you possibly work on your own work, but please remember to continue to check in with students every 10 minutes.

If, for some reason, you cannot attend one of your shifts, you are responsible for finding another tutor who can cover your shift for you. Make sure you know what forum your class uses for finding replacements (Piazza, email, etc).

You are welcome to help students with the material on your own time. However, you can only bill hours that you spent during your scheduled tutoring shift, in the appointed location.

Reflections on Tutoring

As you may know from your own experience learning CS, finding the right way to approach bugs can be difficult because they can be due to anything from a flaw in the design of the algorithm to a bug in a library function. You are also dealing with a domain in which there is rarely just one right answer to a problem, so you as tutor must be prepared to adapt your help to the student's approach, rather than encouraging the student to use your own approach. You should also be prepared to deal with different levels of problems, from students who feel like they've done most of the work but just need help tracking down a bug to students who are not sure where to start a problem.

Following all of our recommendations will be difficult to do in practice! You may find it helpful to think of each tutoring shift as a training session for you, as you strive to improve not just your tutoring skills, but your own problem-solving skills. You may find it helpful to focus on a particular skill during a shift to help it become more natural.

There are many possible approaches to tutoring, but one option is the following workflow:

- Introduce yourself
- Ask the student to describe the part of the homework they're working on:
- Ask the student to describe what they want help with.
  - If they don't know how to get started, ask them to describe the problem in detail:
    - What are the goals of the problem?
    - What are the inputs?
    - What are the outputs?
    - What is their relationship?
    - Can they run the algorithm by hand on a small example?
    - Is there a part of the problem that they could write code for? (and worry about the rest later?)
    - Can they describe the algorithm in words?
  - If they have a syntax error, ask them:
    - What line is the syntax error is on?
    - What does the text of the syntax error mean?

    - What does the internet suggest about how to fix this syntax error?
    - What have they tried to fix this syntax error?
  - If their code doesn't work, ask them:
    - What evidence do they have that their code doesn't work?
    - What test case doesn't work and what incorrect behavior or output results?
    - Could they come up with a simpler example that demonstrates the error?
    - Could they walk through their example that doesn't work:
      - by hand?
      - with a debugger?
    - What lines of code might be producing the bug?
    - Why hypotheses do they have for what might be causing the problem?
    - How can they test these hypotheses? (e.g. writing new test cases, adding print statements, using a debugger)

Teaching Good Programming Habits

When the opportunity arises, please try to encourage students to follow good programming habits. Here are some that might apply.

When writing any non-trivial program, write it in stages. Write small pieces of code, compile them, test them, make sure they work correctly, and then repeat. Encourage students to write documentation and tests before they write the code.

Encourage students to use good style, especially in later courses. The basics of good style include writing comments, using clear indentation, using clear variable and method names, and avoiding duplicated code.

When debugging a program, be sure to understand the problem before trying to fix it. Trying a sequence of semi-random changes can quickly trash a program that was nearly correct.

Some General Tips

When faced with error messages, try to help students read and understand the messages rather than jumping immediately to fixing the code. This approach may help students handle similar issue on their own the next time.

When you see a logic error in a student's code, try to help students trace values through the code so that they can discover the error themselves. Try lots of different examples, not only one, to better understand the bug. Be sure that the student thinks about what answer to expect for each example before trying it.

As you help a student find an error, try to point out the clues that you notice. Learning to recognize the clues will help the student long into the future.

Often, bugs are the result of invalid assumptions about the program input. Try to get students to state and examine their assumptions. For example, suggest some print statements for the program, and ask the student what they think the program will print. Then, run the program and compare the student's guess with what was actually occurring.

Never type for a student on their keyboard. Students should always do things for themselves.

If you find that the student is just transcribing what you're saying, you're probably giving too much advice.

Thank you for your valuable contribution to the CS department! Grutors have a huge impact on how students relate to CS, and your commitment to tutoring helps us provide them with the best experience possible.