

# CS1 Students Speak: Advice for Students by Students

Brian Hanks  
Computer Science Info. Sys.  
Fort Lewis College  
Durango, CO 81301 USA  
hanks\_b@fortlewis.edu

Laurie Murphy  
Comp. Sci. & Comp. Eng.  
Pacific Lutheran University  
Tacoma, WA 98447 USA  
lmurphy@plu.edu

Beth Simon  
Comp. Science & Engineering  
U. of California, San Diego  
La Jolla, CA 92093 USA  
bsimon@cs.ucsd.edu

Renée McCauley  
Computer Science  
College of Charleston  
Charleston, SC 29424 USA  
mccauleyr@cofc.edu

Carol Zander  
Computing & Software Sys.  
U. of Washington, Bothell  
Bothell, WA 98011 USA  
zander@u.washington.edu

## ABSTRACT

We collected advice on how to succeed at learning to program from 164 CS1 students at 3 institutions during a “saying is believing” intervention designed to encourage a growth mindset. More students gave general advice (63%) than programming-specific (23%) or attitudinal advice (34%), despite being prompted to encourage future students to develop a growth mindset toward programming. Advice categories and quotes offer educators insights into student beliefs and practices and suggest a framework for considering how best to advise students. We discuss the implications of students offering advice to other students and provide a handout of representative advice intended for distribution to students in introductory programming courses.

## Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]:  
Computer Science Education

## General Terms

Human Factors

## Keywords

advice, attitudes, mindsets, self-theories, success factors

## 1. INTRODUCTION

As educators, we frequently give advice to students in our introductory programming classes about how to be successful. We feel comfortable stressing the importance of going to class, keeping up with assignments, and seeking help when the material is unclear. But, we don’t know if students consider this advice helpful or if they don’t find it credible because it is coming from us. We could more effectively advise

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*SIGCSE’09*, March 3–7, 2009, Chattanooga, Tennessee, USA.  
Copyright 2009 ACM 978-1-60558-183-5/09/03 ...\$5.00.

students if we knew what kinds of advice students find valuable, and who that advice should come from.

As part of a study on self-theories [7, 9], we asked students nearing the end of their introductory programming course to offer advice for students who might take the course in the future. Advice was solicited in the context of a “saying is believing” experiment designed to encourage students to advocate a growth mindset—the belief that hard work and persistence, rather than innate talent and ability, are fundamental to success. Although statistical results from the experiment were inconclusive with respect to shifting students’ mindsets, their advice gives educators unique insights into student thoughts about what it takes to succeed at programming.

Students from three institutions were involved in the experiment. Their advice fell into three broad types: general study advice, programming-specific advice, and attitudinal advice. In the following sections we describe related work, outline the study methodology, and describe results of our qualitative analysis. Finally, we discuss implications for teaching and offer educators a one-page handout of representative advice from students to distribute in their classes.

## 2. RELATED WORK

Over the past 30 years, psychologist Carol Dweck and others have found links between mindsets about intelligence and academic success (for a summary see Dweck [4]). Those who hold intelligence to be fixed tend to avoid challenges and perform worse academically than those who believe their intelligence can increase with effort and hard work. Mindsets have been suggested as particularly relevant for CS education for several reasons, including the inherently challenging nature of learning to program and cultural and gender-related attitudes toward CS [7].

Our intervention was based on a study conducted by Aronson et al. [1] with freshmen students at Stanford University. They implemented a “saying is believing” intervention with the goals of improving the educational experience for students in general, and of diminishing stereo-type threat among African American students in particular. They found that students, especially African American students, who related the benefits of a growth mindset to their own experiences obtained higher grade point averages and reported greater enjoyment and engagement in the academic process.

Table 1: Advice Categories

<p><b>General Study</b></p>	<p><b>Analogies:</b> Making analogies to other domains such as foreign languages or music  <b>Attend Class</b>  <b>Consider Alternatives:</b> Ask “What if?” questions, take different approaches  <b>Group Work:</b> Work/study/learn with others  <b>Learn from Mistakes:</b> General, no reference to process, compiler, etc.  <b>Learning Approaches:</b> e.g., learn basics first, build a strong foundation  <b>Listen/Pay Attention/Concentrate</b>  <b>Pay Attention to Feedback</b>  <b>Practice:</b> General advice about practicing, trying on your own  <b>Questions/Help:</b> Ask for help in person or by email  <b>Resources:</b> Use resources such as Google, Java API, reference books  <b>Review:</b> Review your code, notes, and the textbook  <b>Stay on Top of Things:</b> Keep up with the work, programming is cumulative  <b>Take a break</b>  <b>Test Taking</b>  <b>Textbook:</b> Any reference to the textbook  <b>Understand Expectations:</b> Know what the professor expects</p>
<p><b>Programming Specific</b></p>	<p><b>Conceptual Understanding:</b> Getting a big picture, forming a mental model, etc.  <b>Details are Important:</b> Pay attention to syntax, keywords, punctuation, etc.  <b>Learning Approaches:</b> Learn syntax first, etc.  <b>Learn from Mistakes/Errors:</b> Bugs and compiler messages are learning opportunities  <b>Practice:</b> Experiment/tinker, introduce bugs intentionally, etc.  <b>Problem Solving – Other:</b> Look at the big picture, program from examples, etc.  <b>Problem Solving – Work Incrementally:</b> Approaches that specifically mention programming in small chunks or steps  <b>Programming Style:</b> Use good identifiers, indentation, comments, etc.  <b>Testing/Debugging/Fixing Errors:</b> Tracing, diagnostic statements, test cases, etc.  <b>Use Tools:</b> Use the compiler, debugger, IDE, or other tools</p>
<p><b>Attitudinal</b></p>	<p><b>Avoidance:</b> Don’t take the class in the first place  <b>Be Confident</b>  <b>Don’t be Frustrated; Be Patient</b>  <b>Don’t be Overconfident</b>  <b>Don’t be Scared/Intimidated</b>  <b>Don’t Panic:</b> Stay calm  <b>Focus Long Term:</b> Keep a long term perspective  <b>General Encouragement:</b> Keep a good attitude, focus on learning  <b>Have Fun/Enjoy Learning</b>  <b>It’s OK to be a Novice:</b> Understand you’re a beginner, learn at own pace  <b>It’s OK to be Confused:</b> Confusion is a natural part of learning to program  <b>Persevere:</b> Don’t give up; there is always a solution</p>

Space does not allow us to give an adequate overview of the vast research into factors that influence student success at learning to program. Research related to advice in this study includes investigations of student behaviors (e.g., learning strategies [2]) and psychological factors (e.g., self-efficacy [10]). Reflections from upper-level CS students about how they manage to get unstuck [6] and phenomenographic investigations of instructors’ beliefs about student success [5] also mirror many of the behaviors, techniques, and attitudes articulated by novices in this study.

There is also a wealth of general advice on how to succeed in college and in CS available on the Internet from professors and universities. This study differed in that it solicited advice from CS1 students for other CS1 students.

### 3. METHODOLOGY

Advice from students in this study was collected during an intervention designed to encourage CS1 students to adopt a growth mindset toward programming [9].

### 3.1 Data Collection

Students from three institutions participated in this study. Institution P is a small, private, liberal arts university located in the United States. The introductory Java programming course is required for computer science, computer engineering, mathematics, and physics majors. A few students from other majors take the course as an elective. Seventeen students from the fall 2007 term (taught by one of the authors) provided advice.

Institution A is a larger university located in the United Kingdom, in which students are admitted directly into the computer science major. Depending upon the amount of previous programming experience, students take one of two introductory Java programming courses, which are taught by separate instructors. A total of 63 students in the Fall 2007 sections of these two courses gave advice.

Institution U is a large North American research-extensive university. Subjects in this study were enrolled in a CS1 course specifically for engineering students (not computer science majors). Eighty-four students offered advice.

Students in these courses were asked to provide advice, based on the following question:

What advice would you give a beginning programmer in [COURSE NUMBER] to help them cope with the challenge of learning to write and debug Java programs? Be sure to emphasize to them how to grow their programming intelligence through dealing with programming challenges.

Students at Institution P received a small amount of extra credit for answering the question. At Institution A, students did not receive any credit for their answers (given during a laboratory period). The question was part of a required laboratory assignment at Institution U.

### 3.2 Data Coding

Three of the authors independently read the responses using a grounded-theory approach [3] identifying 40 categories of advice given by students. Three themes emerged through discussion of these categories: general study advice, programming-specific advice, and attitudinal advice, as shown in Table 1.

Three authors independently coded each response to determine whether or not it included each of the three advice types. The number of coders who marked each type of advice for each response could range from 0 to 3. For example, a response might have received a general advice count of 0 (none of the coders believed it contained general advice), a programming-specific count of 3 (all coders believed it contained programming-specific advice), and an attitudinal advice count of 2 (only 2 of the coders believed it contained attitudinal advice).

## 4. RESULTS

Table 2 shows the percentage of responses for each possible count, for each advice type. The last row shows the level of complete inter-rater agreement for each of the three types of advice. We attained inter-rater agreement of 83% overall, where agreement meant all three coders agreed about the presence or absence of a particular advice type within a student response.

As shown in Table 2, a majority of students (63.4%) gave general advice, while relatively fewer students gave programming-specific (22.6%) or attitudinal (33.5%) advice.

### 4.1 General Advice

General advice was the most ubiquitous and included practicing, learning from mistakes, seeking assistance when stuck, keeping on top of the material, and attending class.

Advice about the importance of practicing was among the most common and is a category under both the general and programming-specific advice themes. General practice advice was often brief and lacked any programming-specific details. More detailed practice advice was considered general if it was still meaningful when another discipline or skill (such as mathematics) was substituted for programming.

Examples of more insightful general practice advice suggested that students should:

practice their skills with a large variety of different programs that would require them to use a lot of different forms of the program [U38]

**Table 2: Coded Advice Distribution**

Number of Coders	General	Program Specific	Attitudinal	Total
3	63.4%	22.6%	33.5%	39.8%
2	9.7%	9.8%	5.5%	8.3%
1	9.2%	10.4%	5.5%	8.3%
0	17.7%	57.3%	55.5%	43.5%
0 and 3	81.1%	79.9%	89.0%	83.3%

us[e] examples of other programs that experienced programmers have done and try to do one similar. [U44]

One general piece of advice critical in learning to program is keeping up in the course, particularly because the material is cumulative. Students commented on this saying:

keep on top of the notes and chapters is important because the next lecture keeps building from what you just learnt so don't fall behind. [U35]

if you fall behind in programming, it's very hard to catch up. [U81]

### 4.2 Programming-Specific Advice

Only 22.6% of the students offered any programming-specific advice. We were somewhat surprised by this, since much time is spent doling out such advice, and because we believe that learning to program calls for targeted advice.

*Advice about tinkering:* Although “tinkering” has been reported in a negative light with regard to debugging [8], we found instances of programming-specific practice advice that described ways students should deliberately tinker in order to learn from practice. Examples include:

I highly recommend trying things. “What if I do \_\_\_\_\_”, then test it. Often I have to repeat this 5 – 6 times ... [U23]

Copy code given in lectures and alter it, change it to a subject you have an interest in. [A23]

Make deliberate mistakes to see output [A36]

*Advice about working incrementally:* A variant of programming-specific advice discussed problem solving steps or programming in small chunks.

[Y]ou'd better do the programs step by step. It means don't write the whole program and run it then try to debug it, ... [P4]

Also, breaking the tasks/programs into smaller parts or steps ... make it easier to deal with. Thus, you are not always overcome with the size of the program you are dealing with. [U39]

*Advice about conceptual understanding:* Some of the most high-level advice about programming suggested visualizing or developing a mental model of the program in execution:

I often notice my colleagues grow frustrated with the compiler during labs ... as they expect the computer [to] “think” like a person, and be able to understand what they want without explicit instructions. Those new to programming should realize that computers have no form of derived intelligence ... [U34]

### 4.3 Attitudinal Advice

Attitudinal advice was given by 33.5% of the students. This number may be artificially high since students were directly prompted to mention developing a growth mindset towards their computing studies.

Many classes students take at the university are advanced treatments of topics studied in high school. CS1 is, for many students, a novel experience – and some felt compelled to emphasize to their peers that being a novice is OK:

Also, realize that [THIS COURSE] is an introductory class. If ever you feel frustrated, just know that everyone else has been there too. [P09]

The main thing I would suggest is to make sure you ask questions in lectures and not to worry about looking stupid for not knowing. Everyone is at different levels, there are bound to be other people that know less or the same as you. [A47]

Some responses could not be considered advice at all, although a few of those were clearly attitudinal:

Java is like an onion. There's loads of layers and it really stinks!

### 5. DISCUSSION

Collectively, the 164 CS1 students in this study advised their peers that to be successful in CS1 they should focus their efforts on three separate yet complementary areas: general study, programming-specific, and attitudinal practices. The accompanying handout presents actual student advice organized around these three themes. Given the relative lack of programming-specific and attitudinal advice from students in this study, we encourage instructors to highlight these in particular.

Learning to program is highly complex. Therefore it is not surprising that many novices in this study had difficulty articulating useful programming practices, even though it is likely their teachers conveyed such information to them. It is possible that programming practices based on the authentic experiences of other novices may resonate more with beginners than advice given by experienced instructors.

Our interest in attitudinal practices is grounded in self-theories research [4] which has shown that students are more academically successful when they adopt a growth mindset. Even though students in this study had attended a short lecture detailing the importance of growth mindsets and were prompted to encourage others to “grow their programming intelligence through dealing with programming challenges”, only 34% gave attitudinal advice. Based on self-theories research and our own anecdotal experiences as instructors, we recommend that students be explicitly encouraged to adopt a growth mindset toward programming.

We have provided an appendix of actual advice from students in this study for instructors to use as a handout in their introductory programming courses. While we believe this advice from other students should prove helpful to beginning programmers, the act of giving advice may be even more valuable. According to “saying is believing” theory [1], advocating a position in one’s own words increases the likelihood that a person will internalize and believe that position. A student who reflects on what it takes to be successful at programming and offers advice such as “read the textbook”,

“work incrementally”, and “don’t give up”, may make a more conscious effort to follow this advice as well.

Finally, asking students to express beliefs about what contributes to their success offers instructors an additional resource for guiding students. Comparing their responses with those given by students in this study may reveal missing or weak areas in need of attention, or offer new pearls of wisdom to add to the representative advice list.

### 6. ACKNOWLEDGMENTS

We thank Sue Fitzgerald for her leadership, and Paul Carter and Lynda Thomas for collecting data. This work was supported in part by NSF DUE grant #0647688.

### 7. REFERENCES

- [1] J. Aronson, C. Fried, and C. Good. Reducing the effects of stereotype threat on African American college students by shaping theories of intelligence. *Journal of Experimental Social Psychology*, 38(2):113–125, 2002.
- [2] S. Bergin, R. Reilly, and D. Traynor. Examining the role of self-regulated learning on introductory programming performance. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 81–86, 2005.
- [3] J. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.
- [4] C. S. Dweck. *Self-Theories: Their role in motivation, personality and development*. Taylor & Francis, 1999.
- [5] P. Kinnunen, R. McCartney, L. Murphy, and L. Thomas. Through the eyes of instructors: a phenomenographic investigation of student success. In *ICER '07: Proceedings of the third international workshop on Computing education research*, pages 61–72, 2007.
- [6] R. McCartney, A. Eckerdal, J. E. Mostrom, K. Sanders, and C. Zander. Successful students’ strategies for getting unstuck. *SIGCSE Bull.*, 39(3):156–160, September 2007.
- [7] L. Murphy and L. Thomas. Dangers of a fixed mindset: Implications of self-theories research for computer science education. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 271–275, 2008.
- [8] D. N. Perkins, C. Hancock, R. Hobbs, F. Martin, and R. Simmons. Conditions of Learning in Novice Programmers. In E. Soloway and J. C. Spohrer, editors, *Studying the Novice Programmer*, pages 261–279. Lawrence Erlbaum Associates, 1986.
- [9] B. Simon, B. Hanks, L. Murphy, S. Fitzgerald, R. McCauley, L. Thomas, and C. Zander. Saying isn’t necessarily believing: Influencing self-theories in computing. In *Proceedings of ICER 2008*, 2008.
- [10] S. Wiedenbeck. Factors affecting the success of non-majors in learning to program. In *Proc. of the 1st Intl. Computing Education Research Workshop (ICER 2005)*, pages 13–24, 2005.

# How to Succeed in Your First Programming Course

These quotes are from students who were nearing the end of their first programming course. They were asked to provide advice to future students (such as you!) about how to be successful in this course.

Their advice falls into three categories: general advice, programming-specific advice, and attitudinal advice. It is important for beginning programming students (like you!) to recognize all three of these areas as part of good practices that, when employed regularly and thoughtfully, can contribute to success in computing classes. Do your practice, habits, and behaviors regarding your programming course include aspects from all three of these categories? If not, these students would say it should.

These students are similar to you. They may not have had any programming experience before this course. They might feel that everyone else knows more than they do. They might have struggled with and been frustrated by the course material. Their advice is based on this relevant experience.

## General Advice

*Make the most of class:* “attend the class regularly since that this course is not an easy course and needs lots of patience. Clear your doubts immediately. Pay attention to the lecture. Don’t hesitate to ask the instructor if you don’t understand any of the presented item.”

“I think the most important thing is to get help when you need it. In high school I didn’t really need to get help with anything because it wasn’t that hard and I felt that if I had to get help then I wasn’t smart”

“Firstly, they have to stay on top of the topic, if you fall behind in programming, it’s very hard to catch up. As well if you fall behind, it’s not like you can forget that certain topic, the whole class is a cumulative topic, you need to know how to do something first before you can move on to the next topic. Secondly, you have to do more than just listen in class”

*Learn from mistakes:* “The key to understand is using each error as a stepping stone and using each error to learn. Figuring out WHY the error is present will force you to understand the reasoning”

“Everybody makes mistakes – learn from them”

*Learn from others:* “Working with other people also helps because what one person doesn’t understand, another person might know how to do.”

“it is important that you take advantage of your professor’s office hours to ask questions and seek assistance”

## Programming Specific Advice

*You have to program to learn:* “After solving a problem, ask yourself how you could change it and play around with it. Experimenting with code is the best way to learn.”

“Experiment by yourself. By playing around you figure out and remember how to do things.”

“Make deliberate mistakes to see output”

“Do some research if a program doesn’t run. Use the debugger.”

“My advice would be to pay attention to details and study examples. Make sure you understand why each character (the brackets [ ], the semicolon, the triangular brackets <>) are there. You could try to remember what each error message means, so you could know what you do wrong the next time you see the same error message.”

*Have a plan:* “1. read through the descriptions of the program you are going to do, make sure you know what you are suppose to do. 2. make a flow chart to help you get clear about the program. 3. write the program step by step. 4. don’t be frustrated if you cannot compile the program, check each line carefully and think of what kind of mistake will be made. 5. if you cannot find out your mistakes, go and find someone to help you, or discuss with your friends.”

“Also, when writing the program test each block of code at a time. If errors are to occur then the programmer would know which block to look in for the errors.”

*It’s not over ’til it’s over:* “Test your programme with different inputs in order to make sure it produces the correct output. Sometime the programme might run correctly for a few inputs only.”

## Attitudinal Advice

*It’s OK, you are just a beginner:* “The main thing I would suggest is to make sure you ask questions in lectures and not to worry about looking stupid for not knowing. Everyone is at different levels, there are bound to be other people that know less or the same as you.”

“If ever you feel frustrated, just know that everyone else has been there too.”

*Hang in there:* “Don’t give up! Ask tutors for advice, read books on the topics. Re-read parts you don’t understand. Don’t panic! Panicking is the worse thing you can do.”

“As long as one has the desire to learn, they will eventually understand, slowly but surely”

*Mistakes are good:* “they should not get upset or feel inferior but try to do another new program. the more programs they do and the more mistakes they do, the more they learn.”

*Don’t be intimidated:* “do not be afraid to ask questions in class as other students in the class may be wondering about the same thing.”

“make sure you ask questions in lectures and not to worry about looking stupid for not knowing.”

*Programming can be fun:* “Learning something new is a challenge, but it is also an opportunity. Programming is fun when you start to know it, especially when you can run your program successfully.”