

Computer Programming Success Strategies

Many students work hard, yet still don't do well in computer science courses, because they're not using their time well. Here are some strategies for success in this course

Strategies for studying for quizzes and the final exam:

Quizzes and the final are primarily designed to test your ability to program in Java. Thus there is no shortcut to preparing for them, other than learning how to program using all the features of Java covered so far in the course. All of the strategies discussed on this page and other resources of this course are designed to assist you in achieving that goal. There are no shortcuts that will allow you to do well in this course without learning its material.

Often students feel they can do well on the assignments, but that the quizzes are much harder. In fact, in most ways quizzes are significantly easier than assignments. However, on a quiz you do not have a computer to help you, and many students rely far too much on the ability of the compiler to catch errors.

Quizzes are designed so that other factors should generally not make them harder than assignments. Quizzes are designed so time pressure is not a major problem for most students, most of the time. They are also designed so that they do not require detailed knowledge of the Java APIs or seldom used language details, though they do reflect knowledge of Java features that you can expect to pick up through the normal course of reading the text and lecture notes and doing assignments and additional programming projects of your choice.

A leading scientific study examined a great many factors to see what most contributed to student success in college. The **three leading factors contributing to success** were:

- **Time on task:** Ok, so this one is obvious. But it's hard to put in time studying if you're not enjoying it, because you're not succeeding, because you're not studying effectively and taking advantage of the other factors contributing to success.
- **Studying with other students:** Many students rate social life as the most valuable parts of their college experience, but they often fail to extend this to their studies. Studying together can be more fun, and more productive too. You can really relate to each others learning needs. And when you help a friend, the material becomes more lively in your own mind.
- **Involvement with faculty:** Most faculty wish students would seek their help more often. They have a deeper and broader perspective on the material and learning process, and an enthusiasm for the material, which they are eager to share with students. Naturally you learn more, and faculty enjoy working with you more, if you make a good effort to learn what you reasonably can by reading and attempting exercises on your own. Then please do come with any questions you are stuck on.

The most consistent result of research on learning is this: **Learning happen when you are solving problems.** Passive learning, such as listening to a lecture or reading a book with few thoughts of your own, is of little value other than as preparation for **active learning.** Your brain quickly flushes most information unless it is used right away to solve problems. **Active learning means giving assignments and class exercises your best effort. It also means doing your text's (or another text's) self-review questions, exercises, and programming projects (or projects of your own if you prefer) until you can apply the material you have learned.**

On the other hand, **doing assignments is no substitute for reading the text and studying lecture material.** By doing lots of debugging and trying lots of possibilities you may eventually succeed in getting small programs to work with little attention to the book or class material. But you will not become a good programmer this way, or do well on quizzes and the final. To be a good programmer (or succeed as a computer science student) you must know the underlying structure of your language and the principles involved in its design. Otherwise, you will not know what the possibilities are for structuring programs unless you have already done a very similar program, you will make many more mistakes than necessary, and you will sometimes have very great difficulty debugging your code. Though you may eventually figure out some aspects of the underlying principles without studying them directly with the help of a book or class material, it is much easier and more reliable to study the principles directly with these aids.

When you hear a term you don't know, write it down. Look it up after class in the appendix of the text and/or another reference book. If you still don't understand, ask a friend or a member of the course teaching team. .

Don't procrastinate. Yes, everyone does at times, and sometimes it seems to do no harm. But in this course there are two special reasons to avoid procrastination:

- You can't tell how close your programming assignment is to working until it works completely. There's no way to know how many bugs may be lurking inside. You may also get really stumped by a bug and need to get help, which can be impossible to get at the last moment. So the only safe approach is to plan on completing assignments well ahead of time.
- Knowledge in this course is cumulative: you will often have difficulty with the current material if you do not have a firm grasp on what has come before. So if you delay studying the text until shortly before a quiz or exam, you will have more trouble understanding lectures and doing assignments.

Learn to read computer science material like a scientist. It's tempting to read your text like a novel. You see some code or read some statement and think "I sort of see what is going on" or even "I expect this will be clear later" and continue. This is appropriate for a first reading to get an impression of the material, which is a good idea before the class in which it is discussed. But you *must* go back and reread it until you understand it like a scientist. Reread until you feel you understand *why* everything is the way it is. If you don't understand the why of it, you don't understand the underlying concept, and you won't be able to apply the knowledge when you are programming.

Have a strategy for solving programming problems. Write it down, as informally as you wish. For object-oriented programming it is often best to start with class diagrams or cards. (What classes do you need? What do they do? How do they relate to each other?) For any but very simple methods, write pseudo code. Do not start writing code unless you're sure you have a good picture of the whole program, which can't be the case on any but very tiny programs until you are experienced. Professional programmers do the same on large programs (and most programs are very large).

Whenever possible, debug programs in your head, rather than at a computer. This will often save you *lots* of time. Debugging at a computer can be very time consuming (especially if you are not using an IDDE: an Integrated Development and Debugging Environment). And you can't use a computer in exams and quizzes.

Learn to use an IDE. An integrated development environment requires some up-front time, but this is more than repaid later in increased productivity on bigger programs, especially when debugging is required. **But do not rely too much on your IDE to find your errors.** On quizzes and exams you cannot use your IDE. More importantly, to develop more complicated programs you must know your language so well that you can think creatively in it and compose code with only occasional errors.

Seek help when you need it. It's important to try solving problems on your own first, but when you're really stuck, get help from the course teaching team, a friend, or a tutor. (Help with specific code in an assignment from any but the teaching team needs to be acknowledged conspicuously in the assignment and may result in loss of assignment credit.)

Review these strategies often. You may have to overcome deeply ingrained habits to use them. This can be done, but requires repeated attention.