UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS 61A
Spring 2011

Brian Harvey

CS 61A: Structure and Interpretation of Computer Programs
General Course Information

Introduction

The CS 61 series is an introduction to computer science, with particular emphasis on software and on machines from a programmer's point of view. This first course concentrates mostly on the idea of *abstraction*, allowing the programmer to think in terms appropriate to the problem rather than in low-level operations dictated by the computer hardware. The next course, CS 61B, will deal with the more advanced engineering aspects of software—on constructing and analyzing large programs and on techniques for handling computationally expensive programs. Finally, CS 61C concentrates on machines and how they carry out the programs you write.

In CS 61A, we are interested in teaching you about programming, not about any particular programming language. We consider a series of techniques for controlling program complexity, such as functional programming, data abstraction, object-oriented programming, and query systems. To get past generalities you must have programming practice in some particular language, and in this course we use Scheme, a dialect of Lisp. This language is particularly well-suited to the organizing ideas we want to teach. Our hope, however, is that once you have learned the essence of programming, you will find that picking up a new programming language is but a few days' work.

Instructor

Brian Harvey
781 Soda Hall
Phone: 642-8311
Computer Mail: bh
Office Hours:
10:00–11:30 Monday,
3:10–5:00 Wednesday

Office hours are primarily for short questions and administrative problems. I am happy to make appointments for longer periods of time if you feel you need it. Please don't be shy; I'd rather see you as soon as you don't understand something than right before the exam. It's not an imposition—I *like* teaching. Please don't ask me about individual *administrative* problems in lecture; I have to be in my office to have access to my files.

Do You Belong Here?

Math 1A is a corequisite for 61A. (That is, it may be taken concurrently.)

In the past we did not impose any programming-related prerequisites for admission to 61A. However, in recent years we have found that 80% to 90% of 61A students have had significant prior programming experience, and that students without such experience are at a disadvantage. You certainly have adequate background for this course if you are familiar with the idea of *recursion*: a procedure invoking itself as a subprocedure. There is no need for you to be familiar with any particular programming language, although if all of your experience has been in BASIC then you probably haven't used recursion. If you've taken the CS Advanced Placement AB course in C++ or Java, you are certainly ready for 61A.

If you don't feel ready for 61A, we recommend that you take CS 10, which is an introduction to computer science for non-majors including Scratch programming, or CS 3S, a self-paced programming class for people who are pretty sure they will be CS or EECS majors. Both are designed to serve as preparation for 61A. You could then take 61A next semester.

Non-technical students (for instance, ones who prefer Math 16A to Math 1A) should probably avoid CS 61A entirely, and should take CS 10 to satisfy their need for or interest in a programming course.

If you are not strongly interested in computer *programming* at all, but instead want to learn how to *use* computers as a tool, you should consider IDS 110, a course that presents a variety of personal computer software along with a brief introduction to programming.

If you have substantial prior programming background, you may feel that you can skip 61A. In most cases we don't recommend that. Although 61A is the first course in the CS sequence, it's quite different from most introductory courses. Unless you have used this same textbook elsewhere, I think I can promise that you won't be bored. If you're not convinced, spend some time looking over the book and then come discuss it with me. Instead, perhaps your prior experience will allow you to skip 61B or 61C, which are more comparable to courses taught elsewhere. You can ask me about this at my office hours.

Course Materials

The textbook for this course is *Structure and Interpretation of Computer Programs* by Abelson, Sussman, and Sussman, second edition. It should be available in the textbook section of the ASUC bookstore and other local textbook sellers. **You must get the 1996 second edition! Don't buy a used copy of the first edition.** The book can also be read online, at

`http://www-mitpress.mit.edu/sicp/sicp.html`

In addition to the textbook, there is a required reader in two volumes. **Volume 1 contains the current semester's assignments, so you must get a new copy this semester.** Volume 2 contains unchanging reference material for the course, so you can use a copy from a previous semester. You will be able to buy the reader at CopyCentral, 2483 Hearst Avenue.

The course reader includes my lecture notes. What this means is that you should be able to devote your effort during lecture to thinking, rather than to frantic scribbling.

If you haven't used Unix before, you should also read the *User's Guide to Unix on EECS Instructional Facilities* available on the web at

`http://inst.eecs.berkeley.edu/pub/html/Users.Guide/`

We have listed an optional text for the course, *Simply Scheme*, by Harvey and Wright. It really is optional! Don't just buy it because you see it on the shelf. This textbook gives a slower and gentler introduction to the first five weeks of 61A, for people who feel swamped here. Most of you won't need this book.

If you have a home computer, you may want to get a Scheme interpreter for it. The Computer Science Division can provide you with free versions of Scheme for Linux, Windows, or MacOS. The distribution also includes the Scheme library programs that we use in this course. You can dowload copies of this distribution at

`http://inst.EECS.Berkeley.EDU/~scheme`

Enrollment—Laboratory and Discussion Sections

In addition to the lectures Monday, Wednesday, and Friday, the course consists of a discussion/lab section twice per week. (You will probably also use some additional unscheduled lab time to do homework problems.) If you are pre-enrolled, you have been scheduled for a section that meets once after the first lecture (that is, on Monday afternoon, Tuesday, or Wednesday morning) and once after the second lecture (on Wednesday afternoon, Thursday, or Friday morning). Most weeks, the first meeting will be in our laboratory room, 271 Soda; the second meeting will be in the classroom listed in the Schedule of Classes. However, **this week's discussion section will meet in the lab!** Since Monday was a holiday this week, you may have missed the first lab meeting. Don't panic about this; you will make up the material later this week.

The discussion sections are run by Teaching Assistants; each TA will handle enrollment for his or her sections. We anticipate some rearrangements during the second week in response to oversubscribed or undersubscribed sections. If you are not pre-enrolled, you should pick a discussion/lab section, but be prepared to shift if your first choice is full. Please try to be in a definite section by the second week, though, because much of the coursework will be done in groups of two to four students (the number depends on the activity); these groups will be set up by the TAs within each section.

You must have a computer account on the 61A course facility. You must set up your account *this week* because that is how we know who is really in the class. If you are pre-enrolled but do not set up your account this week, you may be dropped from the course. Account forms will be distributed at the first lecture, along with this handout. If you missed the first lecture, see Jenny Jones in 339 Soda Hall to get your account. The first time you log in, you will be asked to type in your name and reg card number, if you have one. Please follow the instructions carefully. You must get your account *and log into it* no later than **4pm Friday** so that we have an accurate class count. (In any case, you should be doing the first homework assignment by then.)

Some of you have personal computers and may want to do the course work at home. This is fine with us, although you'll have to be careful to install the class Scheme library on your home computer, to make your computer's version of Scheme behave like the modified one we use in the lab. In any case, though, you must get a class account even if you intend never to use it.

Please do not sign up for a computer science course just to get a computer account, with the intention of dropping later. (Instead, come see a faculty member to discuss sponsorship of a non-class account for independent study, or you can get a free Unix account from the Open Computing Facility.) Accounts of students who are not doing the course work will be turned off by the third week of classes. Also, if you get a class account and then decide to drop the course, please let me know *immediately* so that we can admit another student.

If you are not pre-enrolled and want to take 61A, you have to add the course using Tele-BEARS. If you are something other than a regular Berkeley undergraduate, then you probably need a signature on a form admitting you to the course. Our past experience indicates that we will probably be able to admit everyone, but we can't make a firm promise until after everyone has picked up their class accounts. Therefore, we will not sign your add form until the *second* week of classes. We'll give you instructions about that procedure later. Meanwhile, you should get your computer account and begin doing the course work because if you *are* admitted you won't want to be behind schedule.

Students ask whether section attendance is required or optional. Our expectation is that you will attend all class sessions, but you are adults and we are not going to police your attendance. However, much of the learning in this course comes from lab activities, and later assignments (including exam questions) may build on those activities. It is up to you to find out what happened at any class session that you miss. If you don't have time for all the class meetings, I'd rather you skip the lecture and attend the lab and discussion!

Information Resources

Your first and most important resource for help in learning the material in this course is your fellow students. Your discussion section TA will assign you to a group of four students, and you will do all course activities with this group. **You are responsible for helping each other learn.**

The class will have a staff of undergraduate Lab Assistants (LAs). Each LA will have scheduled hours to be in the lab. (These hours will be posted on the lab door.) Whenever an LA is in the lab (during scheduled hours or not) you may request that s/he answer questions about the homework or programs (but *not* do them for you).

The instructor and the Teaching Assistants who teach the discussion sections are also available to answer questions. You may drop in during office hours, make appointments for other times, or communicate with us by electronic mail.

For technical questions about the homework or about the computer facility, post your questions to the class wiki (see below), but please *don't* post your homework solutions before the assignment is due. You can also send e-mail to your TA or to `bh` about intellectual questions. If you have an administrative question about a *future* assignment (e.g., "I have

to be out of town..."), send e-mail to your TA or your reader, as appropriate. If you have a complaint about a missing or incorrect grade on a *past* assignment, there is an online complaint form accessible from the course web page.

Solutions to homeworks and labs are posted online each week, in the directory `~cs61a/solutions` and on the course web page, the day after each assignment is due. You should definitely read these! They discuss most problems in some depth, with alternate solutions and suggestions for thinking about similar problems. Project and exam solutions are also posted, two days after the due date of each project, and whenever we finish grading each exam (because the posted solutions include a discussion of common wrong answers, which we don't know until we grade them).

This semester, instead of the venerable Usenet newsgroup we've always used for communication within the class, we are trying a new product called "Piazzza" that Dan Garcia says is the bee's knees. You have to register for an account with them, but even though they ask for your full name, university regulations don't allow us to share enrollment data with outside systems, so feel free to make a name up if you'd rather. Definitely don't tell them your student ID or Social Security Number! You can tell them your class login (cs61a-_____). The registration web page is

`www.piazzza.com/berkeley/cs61a`

You should check regularly for announcements, and this is also the best place to ask technical questions. (If you have an administrative question, you might do better sending email to your TA.)

Please do not send electronic mail to every student individually! That would waste a lot of disk space, even for a small message. Use Piazzza instead. Electronic mail is for messages to individuals, not to groups.

There is a class web page, with online versions of some of the documents we hand out:

`http://www-inst.eecs.berkeley.edu/~cs61a`

The web page for the textbook, with additional study resources, is

`http://www-mitpress.mit.edu/sicp/sicp.html`

The optional *Simply Scheme* text is online at

`http://cs.berkeley.edu/ bh/ss-toc2.html`

There are also web pages for the Scheme programming language:

`http://www.swiss.ai.mit.edu/projects/scheme/index.html`
`http://www.schemers.org/`

Tutoring services are provided by Eta Kappa Nu (HKN), the EECS honors society (345 Soda, `hkn@hkn`), and Upsilon Pi Epsilon, the Computer Science honors society (346 Soda, `upe@cory`).

Additional information to help you in studying, including hints from the course staff and copies of programs demonstrated in lectures, is available on the course web page.

Computer Resources

The computing laboratory in 271 Soda consists of about 35 SunRay terminals connected to a Sun Solaris compute server. This is our primary lab room, although the CS 61A accounts can also be used from any EECS Instructional lab in Soda or Cory Hall.

The lab is normally available for use at all times, but you need a card key for evening access to the lab.

Current UCB students: If you are enrolled in the course, your Cal student ID serves as your card key and will automatically be activated for access to the Soda second floor labs (including entering the building). You do not have to do anything, unless for some reason it doesn't work, then see below.

Concurrent/other students: You can fill out an application and obtain a white card key from 387 Soda Hall (the front desk). The fee is $20.00 cash/check, of which $15 is refundable when you return the card.

During scheduled lab sessions, only students enrolled in that particular section may be in the lab. At other hours, any 61A student may use the lab on a drop-in basis. (The labs on the second floor of Soda Hall are unlocked during daytime hours.) If there are no free workstations, please feel free to ask anyone who is not doing course work to leave. In particular, *game playing is not permitted*. We are relying on social pressure to discourage abuse (such as stealing the chairs or monopolizing a workstation for six hours during prime time to play chess). Therefore, do not feel embarrassed to apply such pressure.

You have a homework assignment *this week* (due Monday) that requires you to work on the computer; part of the purpose of the assignment is to ensure that you know how to create, edit, and print files. **Pick up your computer account this week, whether or not you are pre-enrolled.**

These machines use the Unix operating system, a timesharing system that is quite different from the microcomputer systems you have probably seen elsewhere. The course readers include introductory documentation about Unix and about Emacs, the text editing program we are recommending for your use. (It is one of several Unix text editors; you'll find that everyone has his or her own favorite editor and hates all the others. But the people whose favorite is Emacs are right!) Although the use of Unix is not extensively taught in 61A lectures, it will be extremely worthwhile for you to spend some time getting to know how the system works. Each weekly homework assignment includes a suggested "feature of the week" for you to explore. These are entirely optional, and there is nothing to hand in about them. Do the real homework first, but if you have time, you will enjoy learning about the software tools available here. The lab assistants will be happy to get you started and to show you some of the more advanced tricks of the trade.

The Computer Science Undergraduate Association (CSUA) is presenting introductory Unix training sessions. Details will be announced when we have them.

If you have a home computer, you may wish to use your class account remotely. Look here for information on how to do that:

`http://inst/connecting.html`

Computer Community Spirit

If you live in a dorm or other concentrated student housing, you have already learned that any facility shared by a large group of people is fertile ground for practical jokes. You've also learned that selfishness in the use of common facilities can lead to a lot of bad feeling. Computers are no different. For example, there is only a finite amount of file storage space. If you fill it up with digitized pictures of all your friends, other people can't get their homework done.

In the dorm, people generally have a good sense of perspective about what's funny and what isn't. Filling up your friend's room across the hall with balloons is funny. Filling it up with water balloons is on the edge. Filling it up with epoxy isn't funny at all. But for some reason, some people seem to lose that sense of perspective when it comes to computers. Perhaps it's because the damaged property is intangible; perhaps it's because with a computer you don't have to be physically near the victim. Whatever the reason, try to overcome it. It may be funny if your friend sees an unexpected message when s/he logs in, but it's not funny if s/he can't complete the course work because of deleted files.

The operating system we use provides enough security so that nothing you do will mess up another user by accident if you're minding your own business. It is certainly possible to mess up the system deliberately. Many of you are familiar with the personal computer environment, in which some people consider it a mark of sophistication to write "virus" programs that interfere with other people's computers. You are now entering a different culture, with different values. Our research work, as at any university, depends on collaboration both within our department and with colleagues elsewhere. Our computer systems are deliberately set up to *encourage* collaboration among their users, and that means encouraging easy access to one another's systems. This policy requires some degree of trust among the participants. If you've ever taken anything out of a safe deposit box at a bank, you know that it's possible to design a high-security shared facility, but that the cost is making it a big pain in the neck to use the secured data. Some computer systems are designed to have bank-level security, and everyone will think you're very clever if you figure out how to mess up such a system. Nobody will think you're clever if you mess up the 61A system.

The form you sign when you get your computer account says that it is for your use only, and for course work only. We are not unreasonably strict in enforcing this rule. Nobody minds if you occasionally play a computer game late at night, if it's the kind that doesn't wreck the keyboards or mice through repeated high-speed banging on one button. Nobody will object even if you occasionally bring a friend to play the game with you, or if you write an occasional English paper on this facility instead of the official English Department computers. But if you are asked to give up the workstation by someone who wants to do course work and refuse, that's unacceptable. Remember, you and your fellow students are the ones who suffer from such selfishness; the faculty and staff have other computers to work on.

Network Etiquette

Our computer facility is part of a worldwide network that lets you communicate with other users both by electronic mail and by immediate connection if you're both logged on at the same time. You will find that the Internet, much like amateur radio, is a good way to make friends.

However, please remember that the network is *not* exactly like amateur radio, in that most of the people using the net are trying to get work done and don't want to spend time talking with you. Therefore, please do not send mail or IM requests to people whom you don't know. For example, if your best friend from home went to college somewhere else and you don't know his or her e-mail address, do not ask randomly chosen people at that college to locate your friend for you. (You can send mail to `postmaster` at most sites.)

In particular, please keep in mind that many female users of the network regard unsolicited communication from unknown male users as a form of sexual harassment. (Research has shown that users with obviously female login names do get more than their share of such unsolicited mail.) Even if your intent is not to harass, bear in mind how your communication might be interpreted by the recipient.

The way to get to know people on the net is to join newsgroups. You can subscribe to groups on an enormous range of topics, both technical and recreational. Most participants in these groups will welcome individual communication that's relevant to the topic.

Here are a few rules of newsgroup etiquette: (1) Do not post to a group until you've read it for a couple of weeks, so you'll know what people consider appropriate topics for that group. (2) Do not post messages in which you quote all of someone else's long message and then add "Me too!" at the bottom. (3) Don't be sarcastic. If you're angry, wait until tomorrow to post your message. Remember, too, that the other person isn't necessarily just like you; he or she may be eight years old, or eighty. (4) **Do not** post, mail, or forward chain letters! You will certainly lose your Berkeley computer account and may find yourself under arrest for fraud.

Homework and Programming Assignments

Each week there will be problems assigned for you to work on, most of which will involve writing and debugging computer programs. You'll work on some of these problems individually, and some in groups. These assignments come in three categories:

• **Laboratory exercises** are short, relatively simple exercises designed to introduce a new topic. Most weeks you'll do these during the scheduled lab meetings the first half of the week, in groups of two to four students.

• **Homework assignments** consist mostly of exercises from the textbook; you'll do these whenever you can schedule time, either in the lab or at home. You may be accustomed to textbooks with huge numbers of boring, repetitive exercises. You won't find that in our text! Each assigned exercise teaches an important point. These assignments are included in the course reader. (The first week's assignment is also attached to this handout.) Each week's assignment will be due Monday of the *following* week. You are encouraged to

discuss the homework with other students, but each of you must prepare and turn in your own solutions. (More on this later.)

• **Projects** are larger assignments intended both to teach you the skill of developing a large program and to assess your understanding of the course material. There are four projects during the semester.

Most of the weekly assignments include problems labelled as "Extra for Experts." These problems are entirely optional; do them only if you have finished the regular assignment and want to do something more challenging. There is no extra *credit* for these problems; people who need more credit shouldn't even be trying them, and people who are doing well in the course should be motivated by the desire to learn.

The purpose of the **homework** is for you to learn the course, not to prove that you already know it. Therefore, the weekly homeworks are not graded on the correctness of your solutions, but on effort. **You will get full credit for an entirely wrong answer that shows reasonable effort!** (But you should test your work. If your solution is incorrect, the grader will want to see some evidence that you know it's incorrect.)

Each homework is worth two points for a reasonable effort, zero points for a missing homework or one that seems to show no effort, or **negative four (−4) points** for a solution copied from someone else.

The four programming **projects** *are* graded on correctness, as well as on your understanding of your solution. The first two projects will be done individually; the last two in groups of two students, but some of the work is split up so that each problem is done by one student. You must work together to ensure that both group members understand the complete results. Projects must be turned in both online and on paper (see below).

Copying someone else's work does not count as "reasonable effort"! This includes copying from your friend who took the course last semester as well as copying from other current students. You will get negative credit for copied solutions, and repeated offenses will lead to more severe penalties. If you don't know how to do something, it's better to leave it out than to copy someone else's work.

You should try to complete the *reading* assignment for each week **before** the lecture. For example, you should read section 1.3 of the textbook before the lecture of Monday, 1/24. (Read section 1.1 as soon as possible this week!) You will have five class meetings (three lectures and two discussion/lab sections) to help you understand the assignment. During that time you should begin to work on the exercises, and you should try to complete them in the lab during the second half of the week. If you're efficient, you'll then have the weekend to read the following week's reading assignment.

Each week's homework assignment will be turned in by noon Monday of the following week (or noon Tuesday if Monday is a holiday). There are two ways to turn in assignments: online and on paper. The first week's homework must be turned in *both* online and on paper. (Part of the purpose of this assignment is to make sure you know how to print things; also, **it is from the paper turnin of this homework that the readers make lists of the students they are grading.**) The remaining homework assignments *must* be turned in online, and *may also* be turned in on paper if you would like detailed comments on your work from your reader.

Paper turnin: There are boxes with slots labelled by course in room 283 Soda Hall. (Don't put them in my mailbox or on my office door!) What you turn in should include transcripts showing that you have tested your solution as appropriate. **Keep your graded papers** until the semester is over. You may need them in case a grade is entered incorrectly.

Online turnin: You must create a directory (you'll learn how to do that in the lab) with the official assignment name, which will be something like `hw5` or `proj2`. Put in that directory all files you want to turn in. Then, still in that directory, give the shell command `submit hw5` (or whatever the assignment name is). We'll give more details in the lab.

The programming projects must be turned in online as well as on paper in the homework box; the deadline is also noon on the Monday that the project is due. For the group projects, only one member of the group will submit the entire project for the group.

Everything you turn in on paper must show your name(s), your computer account(s), and your section number. Please cooperate about this; make sure they're visible on the *outside* of the paper you turn in, not buried in a comment in a listing. For online submissions, your name(s), computer account(s), and section number must be included in a comment at the beginning of the file.

Webcast of Lectures

For those of you who miss lectures, or wish to review them, they are available online:

`http://webcast.berkeley.edu/courses`

Lost and Found

When people bring me found items from lecture or lab, I take them to the Computer Science office, 387 Soda. If someone from another class finds something you left in Pimentel, it may end up in the College of Chemistry office, 419 Latimer, 642–5882. Another place to check for lost items is the campus police office in Sproul Hall.

Testing and Grading

If it were up to me, we wouldn't give grades at all. Since I can't do that, the grading policy of the course has these goals: It should encourage you to do the course work and reward reasonable effort with reasonable grades; it should minimize competitiveness and grade pressure, so that you can focus instead on the intellectual content of the course; and it should minimize the time I spend arguing with students about their grades. To meet these goals, your course grade is computed using a point system with a total of 300 points:

3 midterms	3 * 40	120		14 homeworks	14 * 2	28
final		72		4 projects	15,15,25,25	80

There will be three midterms (during the fifth, eighth, and twelfth weeks of the term) and a final. In the past, some students have complained about time pressure, so we'll hold the midterm tests in the evening (check the schedule later in this handout) instead of during the lecture hour. My goal will be to write one-hour tests, but you'll have two hours to work on them. The relatively large number of tests should reduce your anxiety about

ruining your grade by misunderstanding any one question. In general, tests concentrate on the material that has been covered up to and including the week before the test.

This semester, as an experiment, the exams will be **closed book.** The only thing you may bring to the exam, except for a writing impliment, is one page of notes per exam, cumulative – that is, a total of three pages at the third midterm. Generally I prefer open book tests because students tend to study the wrong way for closed book ones, trying to memorize details. The details should be obvious if you understand the ideas; if not, we'll include them with the problem. Please try to study in an open-book spirit!

The reason for this experiment is that I'm working on creating a self-paced version of 61A, and that format depends on being able to generate exams at any moment from batteries of test questions.

In addition to the 300 points listed above, your working group can earn up to six bonus points **if every group member works to ensure the success of every other member,** as follows. For the second and third midterms and the final, every member of your group will get two bonus points if **your original assigned group is still intact** and the person in your group with the *lowest* score on the previous midterm improves by at least five points (for the final, appropriate scaling is done). But you don't get the bonus points if the lowest-scoring student on the previous exam has dropped the course or changed groups. **The intent of this rule is to encourage the group to study together and to take active responsibility for everyone's learning.**

Each letter grade corresponds to a range of point scores:

A+	283–300	A	276–282	A–	269–275
B+	253–268	B	237–252	B–	221–236
C+	214–220	C	207–213	C–	200–206
D+	190–199	D	180–189	D–	170–179

Notice that this scale is nonlinear; the steps are wider in the B range. A typical C– (barely passing) student would be one who gets 60% scores on exams (114 points), 75% on projects (60 points), and 90% on homework (27 points).

If you make the effort to do the assigned work, you will do well on the weekly homework, since those points are awarded for effort rather than for specific correct results. The projects do require correct solutions for full credit. Finally, the tests are meant to be easy for anyone who understands the material; they do not require great creative leaps.

This grading formula implies that **there is no curve**; your grade will depend only on how well you (and, to a small extent, your group partners) do, and not on how well everyone else does. (If everyone does exceptionally badly on some exam, I may decide the exam was at fault rather than the students, in which case I'll adjust the grade cutoffs as I deem appropriate. But I won't adjust in the other direction; if everyone gets an A, that's great. Any such adjustment will happen at the end of the semester, after all the grading is done, and will be based on the total grade distribution. Often a hard question on one exam is balanced by an easy question on another exam, even though we don't try to write either too-easy or too-hard questions.)

If you believe we have misgraded an exam, first be sure that you understand the solu-

tions and grading standards that we'll post online soon after the exam. If your paper was misgraded *according to those standards*, return it to your TA and file an online complaint (from the course web page) explaining your complaint. Only if you are unable to reach an agreement with the TA should you bring the test to me. The TA will carefully regrade *the entire test*, so be sure that your score will really improve through this regrading! By University policy, final exams may *not* be regraded. They may be viewed at times and places to be announced.

If you believe any assignment has been misgraded, or your grade isn't posted within two weeks of turning it in, you should use the online grading complaint form reachable from the class web page. There are grading complaint deadlines for each assignment, to avoid the problem of students two points away from a grade step after the final suddenly discovering that homework 3 wasn't graded. In the case of midterm exams, you should return your exam to your TA as well as entering a complaint online.

Incomplete grades will be granted only for dire medical or personal emergencies that cause you to miss the final, and only if your work up to that point has been satisfactory.

Cooperative Learning Policy

With the obvious exception of exams, we *encourage* you to discuss *all* of the course activities with your friends as you are working on them.

Each student must solve the weekly homework assignments individually; **it is by struggling with the homework problems that you learn the course material!** However, before you develop your own solution to each problem you are encouraged to discuss it with other students, in groups as large or small as you like. **When you turn in your solution, you must give credit to any other student(s) who contributed to your work.** Working on the homework in groups is both a good way to learn and a lot more fun! Although the homework is graded on effort rather than on correctness, if you take the opportunity to discuss the homework with other students then you'll probably solve every problem correctly. Similarly, each student should solve each problem in the scheduled lab activities, but you are welcome to discuss your efforts with your neighbors in the lab.

The first two programming projects will be done individually. The last two projects are larger, and will be done in groups of two, but with each individual student responsible for specific problems within the project. The groups will be chosen by the TA of your discussion section, although we'll consider requests for specific partners or unusual constraints. Your group will turn in *one copy* of each project, with both of your names and logins listed on it.

Parts of some midterm exams will be done in groups of four; this will be explained further when the time comes. These are the groups that are eligible for bonus points for improvement.

Working cooperatively in groups is a change from the traditional approach in schools, in which students work either in isolation or in competition. But cooperative learning has become increasingly popular as educational research has demonstrated its effectiveness.

One advantage of cooperative learning is that it allows us to give intense assignments, from which you'll learn a great deal, while limiting the workload for each individual student. Another advantage, of course, is that it helps you to understand new ideas when you discuss them with other people. Even if you are the "smartest" person in your group, you'll find that you learn a lot by discussing the course with other students. For example, in the past some of our best students have commented that they didn't *really* understand the course until they worked as lab assistants and had to explain the ideas to later students.

If some medical or personal emergency takes you away from the course for an extended period, or if you decide to drop the course for any reason, please don't just disappear silently! You should inform your project partner and your TA, so that nobody is depending on you to do something you can't finish.

Since the textbook exercises are largely the same from one semester to the next in this course, you may be tempted to turn the official published solutions collected by a friend who's already taken the course. Don't do it, for three reasons: First, it's dishonest. Second, the readers will recognize those solutions and you'll get caught. Third, *doing the homework is the main way you learn in this course.* **Read the published solutions** *after* **you struggle with each problem yourself. Reading the homework solutions each week is an excellent way to prepare for the exams.**

Unlike the homework and projects, the tests in this course (except for the parts specifically designated as group parts) must be your own, individual work. I hope that you will work cooperatively with your friends *before* the test to help each other prepare by learning the ideas and skills in the course. But during the test you're on your own. The EECS Department Policy on Academic Dishonesty says, "Copying all or part of another person's work, or using reference materials not specifically allowed, are forms of cheating and will not be tolerated." The policy statement goes on to explain the penalties for cheating, which range from a zero grade for the test up to dismissal from the University, for a second offense.

In my experience, nobody begins the semester with the intention of cheating. Students who cheat do so because they fall behind gradually, and then panic at the last minute. Some students get into this situation because they are afraid of an unpleasant conversation with an instructor if they admit to not understanding something. I would much rather deal with your misunderstanding *early* than deal with its consequences later. Even if the problem is that you spent the weekend in a drunken orgy instead of doing your homework, please overcome your guilt feelings and **ask for help as soon as you need it.**

Lateness

A programming project that is not ready by the deadline may be turned in until noon Tuesday (the day after the due date). These late projects will count for 2/3 of the earned score. No credit will be given for late homeworks, or for projects turned in after Tuesday. Please do not beg and plead for exceptions. If some personal crisis disrupts your schedule one week, don't waste your time and ours by trying to fake it; just be sure you do the next week's work on time.

By the way, if you wait until Sunday to do the homework, you will probably experience both a shortage of available workstations and unusually slow computer response.

Questions and Answers

Q: Is it true that 61A is the weed-out course for wannabe CS majors?

A: No. The CS major is not overcrowded these days, and we have no interest in weeding anyone out. The work in 61A is *not* designed to be especially hard, although it does require learning new ideas. Several years ago, the number of students seeking admission to the major was artificially swelled because people who cared only about getting rich thought that computer programming was the way to do it. These days, it is no longer possible for any idiot to get rich as a programmer, and so those students are now more likely to choose Business Administration, where they belong. The grading policy in 61A is not harsh and is *not curved* as it would be if we had weeding out in mind. However, you may take this course as an opportunity to weed *yourself* out; if you find that you don't enjoy the work, perhaps you aren't a computer scientist at heart.

Q: I am pre-enrolled for this course, and I'm planning to do the homework on my home computer. Do I still have to pick up a class account and log in by Friday to stay in the class?

A: Yes.

Q: I am a transfer student, and I'm pressed for time to fit in all my graduation requirements. I know how to program. Do I really have to take 61A?

A: Yes, unless you have taken this same course elsewhere. 61A is really very different from the usual first computer science course. However, your prior experience may well get you out of 61B, which is more nearly a standard second course. I can approve course equivalents during office hours or by appointment; bring a detailed syllabus with weekly assignments. In unusual situations, students who have taken a *partly*-equivalent course can fill in the gaps with the self-paced CS 47A; ask Dan Garcia about it.

However, you should bear in mind that 61A is the best CS course in the world (because of the text, not because I'm teaching it), and you'll be missing a great opportunity.

Q: I'm not pre-enrolled. How do I know which discussion section to attend?

A: Along with this handout you should have gotten a one-page additional handout that lists sections and their enrollments. Pick one that has room. If the one you pick is too crowded, the TA may ask some people to move. (Midafternoon sections tend to be the most crowded.) With luck, things will settle down reasonably in a week or so.

Q: Why don't we learn some practical language like Java?

A: First of all, Scheme *is* practical. Of the hundreds of languages that have been invented, Lisp (of which Scheme is a dialect) is the second-oldest survivor, after Fortran. It hasn't lasted 45 years by being useless. Second, and more important, the goal of 61A isn't to teach you a language. The language is just the medium for the ideas in the course, and Scheme gets in the way less than most languages because it has very little syntax and because you don't have to worry about what's where in the computer memory. (Next semester you'll learn Java.) Finally, our textbook, the best computer science book ever written, happens to use Scheme; if they'd used COBOL, we'd teach COBOL for the sake of this text.

Q: I need my extension form signed this week in order to satisfy my employer, or my school, or somebody. Can't you let me in early?

A: I'm sorry, but we have to get an accurate class count first. Often enough pre-enrolled people drop (or just don't show up) in the first week to allow us to let everyone in, but we are not permitted to accept extension students until all regularly enrolled undergraduates are in.

Q: What's your advice on surviving this course?

A: Two things: Don't leave the homework until Sunday, and **ask for help as soon as you don't understand something.**

Q: I got the Nobel prize last year, and my uncle is Chancellor of Berkeley. Do I still have to use my class account by Friday to stay in the class?

A: Yes.

Q: I am disabled and need special facilities or arrangements to do the course work. What should I do about it?

A: The Disabled Students Program (DSP, ext. 2-0518) certifies students as having special needs; DSP students are entitled to the necessary accomodations in course arrangements; the DSP office will give you a letter to bring to us. Please take the initiative about letting us know what you need. For example, if you are qualified to take tests separately, that's fine, as long as you ensure that we've worked out the arrangements before the test. The DSP has voice response terminals from which blind students can connect to our computers. **If English is not your native language,** and you have trouble understanding the course materials or lectures for that reason, please ask for help about that too.

Q: I don't like (or have a conflict with) my pre-assigned discussion section. Can I switch?

A: You must negotiate this with the TA of the section you want to switch into. Please try to be settled into a definite section by next week, when the group assignments will be made.

Q: Isn't it unfair that my grade depends in part on the performance of my project partner?

A: Do you complain about courses that are graded on a curve? It's very common to find a course in which your grade is *hurt* by someone else doing well in the course. If you can accept that, you should be much happier about an arrangement in which your grade is *helped* if you can help someone else learn. There's something unhealthy about a society in which competition is considered okay but cooperation is considered weird.

Q: Can I form a group with a student in another section?

A: Generally not. One purpose of the scheduled lab meetings is to ensure that your group can spend some time working together with your TA available to help. If you want to be in the same group with a friend, arrange your schedules so that you can be in the same section. If there's some special reason why you think you should be an exception, negotiate with the TA or TAs involved.

Q: I'm thinking about buying a personal computer. What do you recommend?

A: For this course, and in general for computer science courses at Berkeley, you don't *need* a computer of your own at all; you can work in the labs on campus. If you just want to be able to connect to the campus computers from home, anything with an Internet connection will do. (If you live in certain dorms, there is an Ethernet connection in your room, and having a computer with an Ethernet adaptor will be very handy.) If you want to work entirely within your home computer, you can get STk for PC-compatibles or MacOS X, or Gambit for pre-OS X Macintoshen, at `http://inst.eecs.berkeley.edu/~scheme`. Some of our students, especially the ones with a particular interest in system administration, choose to run one of the free versions of Unix at home, either Linux or FreeBSD. This takes more effort than using commercial systems, but you learn a lot in the process. MacOS X is an interesting compromise, with BSD Unix hidden behind a toy-computer user interface, so you can install the X Window System and then have an environment like the one in our labs while still listening to your iTunes library.

Q: My project partner never does any work. What should I do about it?

A: First of all, try to find out why. Sometimes people give up because they're having trouble understanding something. If that's the problem, see if you can teach your partner and get him or her back on track. Also, try to find out what his or her *strengths* are—how he or she can best contribute to the group's efforts. But sometimes people get distracted from coursework for non-academic reasons. If you can't resolve the problem yourselves, talk with your TA. With your TA's permission, you may fire your partner *between* projects

(not during a project). Your TA will generally allow you to fire a partner who makes no effort to cooperate, but not one who is trying but having difficulty in the course. (If someone in your group insists on doing all the work, that also counts as not cooperating.)

Firing a partner is a last resort, because we won't accept assignments from individuals, so neither of you will get credit for future assignments unless you can join another group. On the other hand, if you do have a problem with your partner, you should be sure to resolve it quickly, because we will not accept hard-luck stories at the end of the semester about how you lost points undeservedly because your partner never did any work.

Q: I'm not pre-enrolled, so I don't have an official discussion section yet. To which grader should I turn in the first week's homework assignment?

A: The one for whichever section you attended during the first week. Once you have an official section, you should turn homework in under that one. Send email to both readers whenever you switch sections!

Q: What should we call you?

A: If you're being formal, or don't like the course, "Dr. Harvey." Otherwise, "Brian." (*Not* "Dr. Brian"!)

Q: I'm having trouble understanding the assignments. I've never had a problem like this in school before. Does this mean I'm not as good a programmer as I thought, or should I just wait a week or two and see if things clear up?

A: Neither. Most Berkeley students found high school pretty easy, and for many of you, this course will be the first real intellectual challenge you've met. You may have come to believe that everything should be easy for you. On the contrary; if you find your courses easy, you're taking the wrong courses! The whole reason you chose an excellent university was to stretch your mind. (If you chose Berkeley for the sake of a prestigious diploma, maybe you should consider majoring in Business Administration.) *There is nothing shameful about asking for help.* Every semester a few intelligent students end up in trouble in this course because they're too proud to come to office hours with questions. If you wait two weeks before you ask your question, by then you'll feel hopelessly behind, because the topics for those two weeks depend on the idea that you don't understand now.

Q: I forgot my password. (Or: It won't let me log in.)

A: Go to the Instructional sys admin staff in 333 Soda, 378 Cory or 386 Cory. Bring your initial class account form or student ID card. Don't ask for another account, especially if you already have some work graded under the old account!

Lecture Outline and Reading Assignments

In the following chart, the readings refer to section numbers in the Abelson and Sussman text. Remember, the reading should be done *before* the week indicated.

week	Monday	Wednesday	Friday	reading
1	**holiday**	functional programming 1/19	1/21	1.1
2	1/24	higher-order procedures UI (Kay) 1/26	1/28	1.3
3	1/31	UI (Kay) recursion and iteration 2/2	2/4	1.2.1–4
		Project 1 due Monday, 2/7		
4	2/7	data abstraction, sequences calculator 2/9	2/11	2.1, 2.2.1
		Midterm Wednesday 2/16, 7–9pm		
5	2/14	hierarchical data interpreter 2/16	2/18	2.2.2–3, 2.3.1,3
		Project 2 due Tuesday, 2/22		
6	**holiday**	generic operators 2/23	2/25	2.4–2.5.2
		GCD: 5pm Monday 2/28, MT1, Proj1, HW1–5		
7	2/28	object-oriented programming 3/2	3/4	OOP (reader)
		Midterm Wednesday 3/9, 7–9pm		
8	3/7	assignment, state, environments 3/9	3/11	3.1, 3.2
		Project 3a due Monday, 3/14		
9	3/14	mutable data vectors 3/16	3/18	3.3.1–3
		spring break		
		Project 3b due Monday, 3/28		
		GCD: 5pm Monday 3/28, MT2, Proj2, HW6–8		
10	3/28	client/server concurrency 3/30	4/1	3.4
11	4/4	streams Therac 4/6	4/8	3.5.1–3, 3.5.5, Therac
		Midterm Wednesday 4/13, 7–9pm		
12	4/11	metacircular eval. mapreduce 4/13	4/15	4.1.1–6
13	4/18	mapreduce analyzing, lazy evals. 4/20	4/22	4.1.7, 4.2
		Project 4a due Monday, 4/25		
		GCD: 5pm Monday 4/25, MT3, Proj3, HW9–12		
14	4/25	logic programming review 4/27	4/29	4.4.1–3
		Project 4b due Monday, 5/2		
		GCD: 5pm Monday 5/16, Proj4a, HW13–14		
		GCD: 5pm Tuesday, 5/10, Proj4b		
		Final Tuesday, 5/11, 11:30am–2:30pm		

Note: *GCD* = Grading Complaint Deadline.

CS 61A Spring 2011 Week 1

Topic: Functional programming

Monday, 1/17 is a holiday!

Lectures: Wednesday 1/19, Friday 1/21

Reading: Abelson & Sussman, Section 1.1 (pages 1–31)

Note: With the obvious exception of this first week, you should do each week's reading *before* the Monday lecture. So also start now on next week's reading, Abelson & Sussman, Section 1.3

Homework due noon Monday, 1/24:

People who've taken CS 3: Don't use the CS 3 higher-order procedures such as `every` in these problems; use recursion.

1. Do exercise 1.6, page 25. This is an essay question; you needn't hand in any computer printout, unless you think the grader can't read your handwriting. If you had trouble understanding the square root program in the book, explain instead what will happen if you use `new-if` instead of `if` in the `pigl` Pig Latin procedure.

2. Write a procedure `squares` that takes a sentence of numbers as its argument and returns a sentence of the squares of the numbers:

```
> (squares '(2 3 4 5))
(4 9 16 25)
```

3. Write a procedure `switch` that takes a sentence as its argument and returns a sentence in which every instance of the words `I` or `me` is replaced by `you`, while every instance of `you` is replaced by `me` except at the beginning of the sentence, where it's replaced by `I`. (Don't worry about capitalization of letters.) Example:

```
> (switch '(You told me that I should wake you up))
(i told you that you should wake me up)
```

4. Write a predicate `ordered?` that takes a sentence of numbers as its argument and returns a true value if the numbers are in ascending order, or a false value otherwise.

5. Write a procedure `ends-e` that takes a sentence as its argument and returns a sentence containing only those words of the argument whose last letter is E:

```
> (ends-e '(please put the salami above the blue elephant))
(please the above the blue)
```

Continued on next page.

Week 1 continued...

6. Most versions of Lisp provide `and` and `or` procedures like the ones on page 19. In principle there is no reason why these can't be ordinary procedures, but some versions of Lisp make them special forms. Suppose, for example, we evaluate

```
(or (= x 0) (= y 0) (= z 0))
```

If `or` is an ordinary procedure, all three argument expressions will be evaluated before `or` is invoked. But if the variable `x` has the value `0`, we know that the entire expression has to be true regardless of the values of `y` and `z`. A Lisp interpreter in which `or` is a special form can evaluate the arguments one by one until either a true one is found or it runs out of arguments.

Your mission is to devise a test that will tell you whether Scheme's `and` and `or` are special forms or ordinary functions. This is a somewhat tricky problem, but it'll get you thinking about the evaluation process more deeply than you otherwise might.

Why might it be advantageous for an interpreter to treat `or` as a special form and evaluate its arguments one at a time? Can you think of reasons why it might be advantageous to treat `or` as an ordinary function?

Unix feature of the week: `man`

Emacs feature of the week: `C-g`, `M-x apropos`

There will be a "feature of the week" each week. These first features come first because they are the ones that you use to find out about the other ones: Each provides documentation of a Unix or Emacs feature. This week, type `man man` as a shell command to see the Unix manual page on the `man` program. Then, in Emacs, type `M-x` (that's meta-`X`, or `ESC X` if you prefer) `describe-function` followed by the Return or Enter key, then `apropos` to see how the `apropos` command works. If you want to know about a command by its keystroke form (such as `C-g`) because you don't know its long name (such as `keyboard-quit`), you can say `M-x describe-key` then `C-g`.

You aren't going to be tested on these system features, but it'll make the rest of your life a *lot* easier if you learn about them.

CS 61A Spring 2011 Week 1 Lab
Wednesday 1/19 afternoon, Thursday 1/20, or Friday 1/21 morning
Try to get as much done as possible, but don't panic if you don't finish everything.

1. (15 minutes.) Start the Emacs editor, either by typing `emacs` in your main window or by selecting it from the alt-middle mouse menu. (Your TA will show you how to do this.) From the `Help` menu, select the Emacs tutorial. You need not complete the entire tutorial at the first session, but you should do so eventually.

(Parts 2–4: 15 minutes.)

2. Use Emacs to create a file called `pigl.scm` in your directory containing the Pig Latin program shown below:

```
(define (pigl wd)
  (if (pl-done? wd)
      (word wd 'ay)
      (pigl (word (bf wd) (first wd)))))

(define (pl-done? wd)
  (vowel? (first wd)))

(define (vowel? letter)
  (member? letter '(a e i o u)))
```

Make sure you are editing a file whose name ends in `.scm`, so that Emacs will know to indent your code correctly!

3. Now run Scheme by typing meta-S in your Emacs window. You are going to create a transcript of a session using the file you just created, like this:

```
(transcript-on "lab1")       ; This starts the transcript file.
(load "pigl.scm")            ; This reads in the file you created earlier.
(pigl 'scheme)               ; Try out your program.
                             ; Feel free to try more test cases here!
(trace pigl)                 ; This is a debugging aid. Watch what happens
(pigl 'scheme)               ; when you run a traced procedure.
(transcript-off)
(exit)
```

4. Use `lpr` to print your transcript file.

Continued on next page.

Week 1 lab continued:

5. (15 minutes.) In the shell, type the command

```
cp ~cs61a/lib/plural.scm .
```

(Note the period at the end of the line!) This will copy a file from the class library to your own directory. Then, using emacs to edit the file, modify the procedure so that it correctly handles cases like `(plural 'boy)`.

6. (20 minutes.) Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

7. (15 minutes.) Write a procedure `dupls-removed` that, given a sentence as input, returns the result of removing duplicate words from the sentence. It should work this way:

```
> (dupls-removed '(a b c a e d e b))
(c a d e b)
> (dupls-removed '(a b c))
(a b c)
> (dupls-removed '(a a a a b a a))
(b a)
```

CS 61A Course summary

You aren't expected to understand this yet, but keep it for reference during the semester and see if it starts to make sense!

ABSTRACTION:
voluntary submission to a discipline in order to gain expressive power

---

1. FUNCTIONAL PROGRAMMING
   - focus: repeatable input-output behavior
     composition of functions to layer complexity
   - hidden: side effect mechanisms (assignment)
     internal control structure of procedures

2. DATA ABSTRACTION
   - focus: semantic view of data aggregates
   - hidden: actual representation in memory

3. OBJECT ORIENTED PROGRAMMING
   - focus: time-varying local state
     metaphor of many autonomous actors
   - hidden: scheduling of interactions within the one computer
     procedural methods within an object

4. CONCURRENCY
   - focus: process separated from serial execution limitations
     coordination among processes
   - hidden: mechanisms for assuring serialization

5. STREAMS
   - focus: metaphor of parallel operations on data aggregates
     signal processing model of computation
   - hidden: actual sequence of events in the computation

6. PROGRAMMING LANGUAGES
   - focus: provide a metaphor for computation
     embody common elements of large groups of problems
   - hidden: technology-specific implementation medium
     storage allocation, etc.

7. LOGIC PROGRAMMING
   - focus: declarative representation of knowledge
     inference rules
   - hidden: inference algorithm

Note: each of these abstractions can be approached "from above," focusing on the view of computing that the abstraction provides, or "from below," focusing on the techniques by which the abstraction is implemented. In the metacircular evaluator we emphasize the view from below, since we've been working all along with the view from above. In the query evaluator we emphasize the view from above, barely mentioning the implementation techniques. In our discussion of object programming both views are used.

EMACS and SCHEME

In Emacs, most commands are invoked with a single keystroke. In this handout, something like C-y is pronounced "control Y" and means that you should hold down the CONTROL key on your keyboard and, while holding CONTROL down, type the letter Y. (Either capital "Y" or lower case "y" is okay.)

Something like M-x is pronounced "meta X" and can be typed in two different ways. On ANY keyboard you can invoke this command with two keystrokes: first type the ESCAPE or ESC key, by itself, and then type the letter X by itself.
On workstation keyboards and on some other terminals, there is a META or ALT key that works like the CONTROL key; it is held down along with the letter.
On some Sunray terminals, the META keys are the diamonds next to the space bar.

Some Emacs commands are of the form C-M-f, "contro“l meta F." If you have a meta key you can hold down CONTROL and META together and type F, all in one keystroke. If not, first type ESCAPE by itself, then type C-f.

Here is how to enter and run a Scheme program using Emacs:

1. You want to create a file containing your program. Give the shell command

    emacs something.scm

The "something" part can be any descriptive name, but the ".scm" should always be used so that Emacs knows this is a Scheme program.

2. Type in your definitions. Emacs will indent each line properly for you.

3. To start Scheme within Emacs, type M-s which will split the screen in two and run Scheme in the bottom half.

4. The first thing you'll want to do is load your entire file into Scheme; this can be done in two ways. One way is to type, in the Scheme window,

    (load "something.scm")

just as you would outside of Emacs. The other way is to switch back to the file window (by typing C-x o) and then type C-c M-l to send a LOAD command for that file to the Scheme window automatically. The fun part comes when you want to change one procedure, without reloading everything:

5. To get from the Scheme window back to the file window, say C-x o (that's two keystrokes, control-X then O, not control-O). This stands for "other window" and can be used to switch the cursor from one window to another.

6. Position the cursor at the beginning of the definition you want to load into Scheme. (Use the usual cursor motion or search commands.) Both of the parentheses that delimit the definition will light up if you're in the right place.

7. Now type C-c M-e to copy that definition into the Scheme process.

8. If the indentation of a definition in your file window gets messed up move the cursor to the beginning of that definition and say C-M-q to fix it.

WORD AND SENTENCE MANIPULATION PROCEDURES

The first chapter of the textbook deals exclusively with numeric data. To allow some variety, with interesting examples that aren't about calculus, we are going to use some additional Scheme procedures that manipulate linguistic data: words and sentences. A word can be considered as a string of characters, such as letters and digits. (Numbers can be treated as words.) A sentence is a string of words in parentheses.

PROCEDURES TO TAKE APART WORDS AND SENTENCES:

FIRST returns the first character of a word, or the first word of a sentence

BUTFIRST returns all but the first character of a word, or all but the first word of a sentence

BF same as BUTFIRST

LAST returns the last character of a word, or the last word of a sentence

BUTLAST returns all but the last character of a word,
or all but the last word of a sentence

BL same as BUTLAST

Examples:

```
> (first 'hello)
 h
> (butlast '(symbolic data are fun))
(symbolic data are)
```

PROCEDURES TO COMBINE WORDS AND SENTENCES

WORD arguments must be words; returns the word with
all the arguments strung together

SENTENCE returns the sentence with all the arguments
(words or sentences) strung together

SE same as SENTENCE

Examples:

```
> (word 'now 'here)
nowhere

> (se 'lisp '(is cool))
(lisp is cool)
```

PREDICATE PROCEDURES

EQUAL? returns true if its two arguments are the same word or the same sentence (a one-word sentence is not equal to the word inside it)

MEMBER? returns true if the first argument is a member of the second; the members of a word are its letters and the members of a sentence are its words

EMPTY? returns true if the argument is either the empty word [which can be represented as ""] or the empty sentence [which can be represented as '()]

MISCELLANEOUS

COUNT returns the number of letters in the argument word, or the number of words in the argument sentence.

ITEM takes two arguments: a positive integer N, and a word or sentence; returns the Nth letter of the word, or the Nth word of the sentence (counting from 1).

Examples:

```
(define (buzz n)
  (cond ((member? 7 n) 'buzz)
        ((= (remainder n 7) 0) 'buzz)
        (else n) ))

(define (plural wd)
  (if (equal? (last wd) 'y)
      (word (bl wd) 'ies)
      (word wd 's) ))
```