Syllabus: CIS 300 Foundations of Computer Science

Brian C. Ladd

Spring 2023

Catalog Description

> Topics include introductions to: propositional logic, sets, and formal languages; mathematical proof techniques; incomputability; and recurrence relations. Fall and Spring. Prerequisite(s): CIS 201 with a minimum grade of 2.0.

Overview

Why do *you* trust computers? That is, when the bank's program calculates the interest that they owe you (or you, them), do you check that it is correct? Why do you accept that it is something that a computer can accurately calculate?

Similarly, when you write your own program that includes, say, sorting some data, if it takes one one-thousandth of a second to sort data for your 100 users, what will happen if your site goes viral and there are 100,000 users? Will it be able to sort in less than a minute? An hour? A day? How could you know?

These questions get at the heart of the theoretical side of computer science: **computablity**, defining the limit of what computers *can* do, and **tractability**, determining what computers can do within a *reasonable* amount of time. Answering these sorts of questions requires a basis in *discrete mathematics* as well as the ability to *prove* facts in mathematics and computer science.

`Foundations of Computer Science` is the gateway to theoretical computer science. Discrete mathematics is the study of *discrete*, or separate, objects. Where calculus studies continuous functions (and hence *real* numbers, $\mathbb{R}$), discrete mathematics studies separate objects, sets of such objects, and functions on sets of such objects (and hence focuses on the set of *integers*, $\mathbb{Z}$ (and, to a lesser extent, *rational* numbers, $\mathbb{Q}$)).

The course uses a *beta* version of a forthcoming textbook, *Connecting Discrete Mathematics to Computer Science* which introduces sets and logic with examples found across computer science. Propositions (truth *statements*) and predicates (truth *functions*) and rules of inference (all from the field of logic) are used to **prove** mathematical theorems.[1] Discrete structures such as sets (finite and infinite), relations, functions, and algorithms are presented and serve as fertile fields to practice proof techniques.

Proof techniques, including direct proof, indirect proof, and induction are the most important material in the course as they underlie all of algorithm analysis and theory of computation. This class will include the proof that there are (*mathematical*) **languages** that cannot be (*mathematically*) **decided** (or **computed**), ever. This fundamental limitation of computability is something that every computer scientist should have in the back of their mind as they look to solve real-world problems.

Remember that the computer you use is *digital*, *binary*, and *general-purpose*. **Digital** (as opposed to *analog*) means that values are chosen from a set (discrete math object) of distinct elements (discrete math objects). **Binary** means that the values are *base-2 digits* or *Boolean values*; Boolean values, also known as *truth*-values, are the set of objects used in mathematical *logic*, propositions, and predicate functions. **General-purpose** means that sequences of bits (another discrete structure) can be combined to *encode* different types of values at different times; encodings of sets in an actual computer provides insight on proving the limitations of computation. The point of this paragraph is to show that though this course is the *foundation* of the **theory** of computer science, theory informs practice across the entire field of computer science.

Calendar Considerations

- The last day to **add/drop**: 2024-01-26
- The last day to withdraw or select S/U: 2024-03-29
- **Winter Recess**: 2024-02-22 – 2024-02-23 (days w/o classes)
- **Spring Recess**: 2024-04-01 – 2024-04-05 (days w/o classes)
- **Last Day of Classes**: 2024-05-10
- **Final Exam**: 2024-05-15 08:00 – 10:00
- Additional Calendar Information: SUNY Potsdam Academic Calendar.

[1] Yes, it is *only* a theorem. Of course in mathematics, being only a theorem means that it has been rigorously proved to be true.

Student Learning Outcomes

`Foundations of Computer Science` introduces the mathematical language and techniques underlying the theory of *complexity* (what can be computed in a **reasonable** amount of time) and the theory of *computability* (what can be computed **at all**). Upon completing this course, students should be able to:

1. **Foundations**. *Relate* mathematical structures (*e.g.* sets, functions, recurrence relations, sequences, alphabets, strings, and formal languages) to computer science concepts (*e.g.* variable types and their encodings, computer functions, recursion, algorithms, and decidability). (Apply)
2. **Proof Techniques**. *Use* direct and indirect proof techniques (*e.g.* chain of implication, proof by contradiction, and both strong and weak induction) to *prove* novel facts in various mathematical and computer science domains (the **Foundations**). (Synthesize)
3. **Undecidability**. *Summarize* the implications, across computer science, of the proof that undecidable languages exist. (Apply)

Note: It would be a very good idea to review this section of the syllabus before exams (especially the final). You can also use it when filling out course evaluations to review what the class taught and what it was expected to teach.

Computer Science Program Student Learning Objectives

The Computer Science department has adopted five **program** learning objectives for students completing any undergraduate major in the department [See https://cs.potsdam.edu for the complete list.] `Foundations of Computer Science` addresses PSLO #1:
Students graduating from the Computer Science department at SUNY Potsdam are expected to be able to

1. *Apply logic and mathematical proof techniques to computing problems, including computability, formal languages, and complexity of algorithms.*

Assessment

This is a 300-level course in your major field. You will be evaluated on how well you can apply the expected learning outcomes. This means your understanding of discrete mathematics and your ability to communicate that understanding in both English (the language of instruction) and mathematical symbols. Your writing and speaking should be clear, concise, and correct.

Philosophy

I have been working very hard since the plague struck to improve student learning in my courses. The grade (A-F, 0-100, or 0.0-4.0) is a very small amount of information. What **you** have learned across a semester distilled down to less than seven bits.

Much of the work in this class is *practice* and you will do some of it *during class meetings*. If we are spending time in class doing problems, that means you will have to spend time **outside** the classroom reading the textbook. There will be content quizzes on these assignments that will be assessed; you should bring questions about the material to class and ask.

One of the goals of lots of practice is for **you** to be able to evaluate your own work. That is, at first, working in groups you will be able to give your classmates constructive criticism on the way to being able to look at your own work critically.

Notice: You (yes, **you**) will *make mistakes*. Rather than being a sign of some sort of failing, mistakes are a sign of growth. When you make mistakes, you prime you brain for learning about the material and will learn the correct approach much more firmly for having corrected the mistake.

One shortcoming of videos, textbooks, and slides is that they are edited to avoid showing incorrect information. This makes it look like a "real" mathematician does not make mistakes. The trick is **editing**, rewriting the messy, perhaps error-prone work that produced the proof into a clear, easy to follow result. **Hint**: That means you should plan on rewriting your notes, your practice in your notes, and, especially, anything you plan on turning in.

Remember that mistakes as you learn are a sign that you are trying and are a place to jump off toward the right answer.

You will be assessed. You will do assessment assignments that you are expected to do on your own *outside of class* to demonstrate what you have mastered. Then, every week or so, there will be a quiz and every several weeks an exam.

In the spirit of mistakes being a *part* of learning, every assessment will be returned with feedback on what you did and did not show mastery of. Some assignments might have a chance for a rewrite.

If you do not turn in a *good-faith attempt* for the initial due date (I will determine whether or not you tried the assignment), you will not be able to participate in the rewrite.

The limited amount of time for rewriting work (on those occasions where it is available) is because the rest of the class keeps right on rolling along. If you need to redo lots of work, you will not be doing the current work. And, that will cascade, putting you further and further behind. My cutoff is set to go to zero when I believe your continuing to work on the old material will adversely impact your current learning.

Feedback

It is important to me (Dr. Ladd) that you *learn* discrete mathematics and its application to computer science. This gives you a strong basis to study *algorithm analysis* and *theoretical computer science*. It also gives you the understanding of mathematics to understand papers and results in the field and keep learning as it moves forward.

Note that learning in an **active** process on the part of the learner (that is you): read and engage with the text and included problems; work problem after problem after problem; work problems in groups; talk about problems and how to work them in class; reflect about working problems.

There is a *theme* in the list of your learning activities: **problems**. Example solutions (in books or class) give students the wrong idea about mathematics. Solutions to real problems are not written down in a clean quarter of a page with no crossing out and no false starts. Problems are solved by looking at definitions and seeing what you know and how close that is to what you need to know. Then, after all the false starts, mistakes, and arrows pointing all over the page, the solution can be *rewritten* in a clean, clear, logical form.

Compare it to an essay: in a book, blog, or newspaper, an article appears, written in correct English with careful word choice, a pleasant cadence, and a strong, logical flow. When *you* write an essay, it does not look like that when you start: research notes on napkins, big X-marks through ~~bole~~ whole paragraphs, and missing topic sentences. The next (and next) draft make it better. You get **good** at writing essays by writing and reading a lot of essays. The same is true about mathematical problems: read good solutions and engage with real problems over and over to build up your intuition. This is a **skill** that grows with practice.

There will also be some *reflection* exercises, small written responses where you will explain what (and sometimes how) you learned the material. Reflection is **very important**: cognitive scientists have shown that reflection greatly improves the fixation of new knowledge for the learner; your answers provide me with a moving snapshot of how you are learning.

Grading Distribution

Dr. Ladd gives grades on a 0.0 – 4.0 linear scale, rounded to the nearest third, just as they are reported on your transcript. Thus getting 30% of the questions correct on a graded assignment means you will get a $4.0 \times 0.30 = 1.20$ rounds to 1.3. The final grade is a weighted average of these scores.

Category	**Percentage**
Practice	**—No Points—**
Participation	0.20
Group Work	
Homework	0.20
Quizzes	0.18
Exams	0.42

Practice Many problems, homework, and worksheets are *practice* for your learning. During class, you are expected to do the problems and be able to explain them when called upon. These may, occasionally, be done in *ad hoc* groups. Remember that the point of these exercises is to **practice** solving problems and **learn** the principles of discrete math.

The most important thing to do when practicing: **Avoid** looking at models, notes, or on-line. Do each problem as if it were a quiz with no notes or book allowed. Recall, if it is successful, is the best way to cement what you know; if unsuccessful, it actually primes the brain to record the answer when you finally look it up. Do *not* give up easily. The harder you work your brain in *practice*, the easier tests and graded assignments will be.

These are practice for a reason: you are learning how to solve the problems. Making mistakes is *encouraged*. These are *formative* assessment, assessment that helps form your knowledge. Learn from your mistakes.

Participation How do *you* learn things that are important to you? A commitment to sustained, disciplined practice is part of it for most things, including both mental and motor skills.

I **invite** you to make such a commitment to `Foundations`, agreeing to: read the textbook; work through examples and practice; join the class on time, every time and be part of the learning community.

Practice and example problems are discrete math *exercises*. Just like running or strength training for sport, mental exercises, done regularly over time, are necessary to develop new skills.

To check the preparation and encourage participation, I will have randomized class roster (that will change when I feel like it) and will direct my questions to specific students during class. Additionally, students will be asked to present their solutions to the problems we are working in class.

Each week, you will be assigned an integer score 0 – 4 based on your level of participation. At the end of the semester, these points will be averaged and count toward your course grade.

Group Work Once or twice a week you will meet with your assigned group to do a group worksheet during the class meeting time. This is another form of practice with the answers coming from the *consensus* of the group.

The point of these exercises is to work together with the other members of your group: talking about the material, **aloud**, following each other's thought process, reading and writing together. The questions in the group work *are* practice but doing them alone is not nearly as helpful as participating in a discussion about discrete math with your peers. If you get the material easily, helping others has been shown to cement it in your memory; if you struggle with some topic, being able to ask your peers for assistance can be less intimidating than seeing the instructor or tutors.

After each group work session, you will get an e-mail with a short follow-up question. This problem *will* be assessed as part of your participation score.

Homework Problems, like the practice ones, will be assigned regularly. Prepared by the practice, these problems will be turned in and graded. They are due at the **beginning of class** on a given day. Upload the answers to Brightspace or hand in a hardcopy when class starts (when the clock strikes the hour; not when you finally get to class; though you plan on being on time, right?). You will have a very short assessment after each group work, something that follows on from the point of the work that will be due the *next* class period (the reason that it is short).

Quizzes About once a week there will be a quiz. Some will be reading checks, some definition practice, and many will practice proof techniques.

Exams There will be three in-class exams and a final exam that is weighted as two exams. Because these are in-class and closed-book, you will need to *memorize* definitions. It would be a good idea to write a page or two of definitions that you can read before starting your homework (or right before an exam if that helps you). Any term used in class or assigned readings is one you should know; you have ample opportunity to ask about definitions and how to use them.

Exam problems will be similar to those in practice and homework, with an emphasis on explaining *how* to solve the problem and explaining *what* the solution means. You will have to generate solutions and proofs and then connect that work to a broader understanding of where the work fits in computer science.

Textbook and Readings

`https://cs.carleton.edu/faculty/dln/book/` is the home of *Connecting Discrete Mathematics and Computer Science* with a complete beta version of the book. It is available in PDF by chapter. You should download the whole thing soon so that you do not have to fret if the network becomes unreachable the evening before a class.

General Rules

Your choices make your fate. Information arms you to make your best decisions and enjoy the best fate. This section describes the class expectations so that you can meet them (or decide not to).

Instructor Expectations of Students

Communication

Read/respond to e-mail. Read/respond to the Brightspace site. These are the two primary means of communication in the class. You should make use of them.

Brightspace contains the course *schedule*. That schedule contains readings, writings, programs, presentations, and homework deadlines appear. If anything is missing or you find discrepencies, ask.

Students are expected to have a copy of the textbooks (if any) and are expected to complete reading assignments *before* the beginning of class the day they are due. In-class participation requires you to have engaged the reading. **Note:** this is particularly true for the original source assignments. If you have not read the assigned portion of the project, you will get nothing out of the discussion.

Students are expected to listen actively in class. They are also expected to take notes in class. These two activities (which are linked) correlate strongly with understanding the material presented. Do not copy slides or board notes directly: digest them and restate them in your own words. Mark where you expect test questions to lurk (sometimes the instructor will even help with this).

Advanced courses often use a version control system (such as `git`) to retrieve and submit source code and programming assignments. This requires you to understand how to use `git`, to understand the difference between the *class* repository (which belongs to the instructor) and *assignment* repositories (which belong to you).

Attendance

American undergraduates are old enough to join the army, vote, and even get married. You are each mature enough to make your own decisions about attending class. Be aware that your decisions have *consequences*.

Students are expected to attend every class. Students are responsible for all material covered in every class meeting. There is no taking of attendance in this class (see 'old enough' in the previous paragraph).

The study of computer science is cumulative; past experience shows a strong correlation between high absences and low grades.

The registrar schedules the final exams every semester; I am not able to move them and do not give early finals. Take-home final exams are due *before* the end of the scheduled exam period (it *will* be accepted early).

Do Your Own Work

Do your own work. That should go without saying but it appears that for some students it bears repeating. Doing your own work means **not** using online resources such as `chegg.com` to look up answers or asking *ChatGPT* (or the generative AI of the moment) to solve problems or design code. Those constitute **academic dishonesty.**

Code you plan to turn in for an individual grade should not be shared with any other student; you should never look at another student's project code, not even for 'guidance.'

I *want* you to discuss assignments and show off results of your playing with programming different things. Discussion at a high-level, even if it includes details on data structures is fine; debugging assistance is also acceptable though the depth of code analysis necessary for debugging skates on thinner ice; directing someone else on exactly what methods a given class should have or the exact format of a programmer-controlled file should take is beyond the pale.

This does *not* apply to group work: in a group project, all members of the team are expected to see and understand all parts of the project. Cross-disciplinary understanding is a highly valued skill in the real world and a major reason for group work in our curriculum.

Artificial Intelligence Note that ChatGPT (or some other new language model) can appear to help you by writing answers for you. Do **not** do this. It is **cheating**. I do not care what *kind* of entity wrote your answers. If it was not **you** then it is not *your* answer and not *your* work.

Turning In Work

Start early, everything takes longer than you think.

As mentioned earlier, reading is to be done *before* class. Homework assignments are also due at the beginning of class so please complete them before class.

Assignments have a due time: a date and a time when they are due. If they are due electronically (they all are), make sure you submit them the expected way by the expected hour. The clock on the receiving computer (Brightspace or the `git` server) determines on-time/late. Do not cut things too close.

Make sure you check the spelling and arithmetic in your assignments. It does not help your grade (or my mood) to turn in sloppy work.

Attitude

Learning is not always easy. Learning is not always comfortable. Students must actively engage the material and be able to ask for help. This does not mean look at the problem for fifteen seconds and throw up your hands because it is 'hard.' It does mean beating your head against it for a bit and then asking me for guidance.

Be respectful. Trust me that I actually have a plan and I know where we are going (okay, I don't *always* know exactly where we are going at every moment but I do have a plan). The readings, the assignments, the lectures, they all go together.

Start early! Everything takes longer than you think.

Student Expectations of the Instructor

Communication

You may expect me to state and keep office hours (you know where my office is, right? ~~301 Dunn Hall~~ Potsdam CS Department Discord). My office hours appear on the course schedule. When I am on Discord, in my voice office, without setting the *Do Not Disturb* setting, you may come on in.

You may expect fairly prompt answers to e-mail (next morning is typical).

You may expect confidentiality: I will not discuss your grades with anyone except my departmental colleagues (and then only to make sure that I am doing my job correctly). I will not share work you have done without your permission.

Attendance

You may expect me to be prepared. This means for class every day and it means with assignments, too. You should expect adequate time to complete an assignment after it is published. You may expect some flexibility (with in the limitations of getting through the course materials) in due dates.

Assignments

Expect assignments to be clear (correct spelling, correct grammar). Expect most sample code to compile and run. Expect assignments to be accompanied by a clear system by which they will be evaluated. Expect me to follow that system.

Students should expect that their work will be evaluated on its own contained merits and not based on how closely it follows the instructor's personal/political views (unless I put parroting my political views into the rubric).

Student Expectations of One Another

Wait, students have expectations of one another? Yes, they should. The most important expectation we should all have of each other is to create a safe learning environment. By safe I mean we should all feel safe to ask questions, to admit mistakes, to try and to be wrong.

Students should be willing to help one another and should all be aware of the limits.

In group work, all students in the group should expect equitable division of labor and should expect to evaluate every member of the group.

Students should expect to evaluate other students' work (and to be evaluated by other students). Both parts of this process have valuable learning outcomes.

Students should expect to feel safe being different. What does this mean in computer science? I actually am not sure. It means respecting each other, evaluating each other in openly communicated terms related to computer science (since that is the topic of the course), and being flexible, accommodating each other whenever possible.

Grading Grievances

When graded work is returned you may have questions about how it was graded. If the question pertains to process (*e.g.,* addition, number of points per subsection), by all means ask me as soon as possible. Typically, if the grade is changed, you will be asked to e-mail me the grade update so that the gradebook is current.

If, however, you feel that your grade fails to reflect the quality of your work (you want to argue that your answer is, in fact, correct, rather than incorrect), take a deep breath and wait. You must wait no less than one and no more than seven days after the work is returned to you to raise the issue with me.

After the cooling-off period, resubmit the work along with a *written* explanation of your concern. This is to assure that your concern gets a cool, thoughtful consideration. If I agree to regrade the work, I will regrade the entire project, replacing the old grade with the new one.

SUNY Potsdam Department of Computer Science Code of Professional Conduct

All members of the Potsdam Computer Science community are governed by the ACM Code of Ethics and Professional Conduct, `https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct`, which we have distilled into our SUNY Potsdam Department of Computer Science Code of Professional Conduct (see below). The department faculty are committed to modeling and promoting ethical and professional behavior for all our students.

Preamble

All members of the ACM, including the Computer Science faculty of SUNY Potsdam, are committed to ethical professional conduct as specified in the ACM Code of Ethics and Professional Conduct. Students, taking courses from the faculty, are bound by our commitment.

All members of the Department are obliged to remind one another to behave professionally. Violations should be reported promptly; however, capricious or malicious reporting of violations is, itself, a violation. When reporting, bring all relevant aspects of the incident to the faculty's attention.

Moral Imperatives

As a Computer Science student I will...

Respect all members of the Department.

1. Be professional in face-to-face and electronic interactions.
2. Be fair so everyone is free to work and learn.
3. Be active in preventing discrimination in physical and electronic spaces frequented by Department members.

Accept and provide appropriate feedback.

1. Avoid starting or spreading rumors.
2. Respect confidentiality.

Be honest, trustworthy, and respect intellectual property.

1. Only take credit for my own work.
2. Respect the privacy of others.
3. Access computing resources only when authorized and report any access risks discovered.

Contribute to society and human well-being.

1. Improve public understanding of computing and its consequences.
2. Consider both the direct and indirect impacts of my actions.

Student Support

Caring Community

I recognize that this is an incredibly stressful time for you, your peers, and our community. Please know that there are resources available to you, both on and off campus, to support you during these very uncertain times. Our excellent Counseling Center staff are available to meet with you; more information can be found on their FAQ page accessed at: https://www.potsdam.edu/studentlife/wellness/counseling-center/coping-covid-19-pandemic/counseling-center-faqs. In addition, information on a variety of on- and off-campus resources can be found our Bear Care site: https://www.potsdam.edu/studentlife/wellness/bear-care. You are an incredibly important member of our Potsdam community; please take care of yourself, and each other.

Every student in this class is a valued individual. If you are struggling with issues outside of the classroom, please know that there are professionals both on and off campus who can assist you. If you need immediate assistance, please contact our campus Counseling Center (with free counseling) at (315) 267-2330 or visit their website. Links to other resources are provided below:

Andrea Waters, Title IX Support Staff & Title IX Core Team

- Sisson 244. (315) 267-2350
- `http://www.potsdam.edu/offices/hr/titleix`

Bias Incident Reporting

- `http://www.potsdam.edu/about/diversity/biasincident`

Center for Diversity

- 223 Sisson Hall
- (315) 267-2184
- `http://www.potsdam.edu/studentlife/diversity`

University Police

- Van Housen Extension
- (315) 267-2222 (number for non-emergencies; for an emergency please dial 911)

Student Conduct and Community Standards

- 208 Barrington Student Union
- `http://www.potsdam.edu/studentlife/studentconduct/codeofconduct`

Reachout (24-hour crisis hotline)

- (315) 265-2422

Renewal House (for victims of domestic violence)

- SUNY Potsdam Campus Office: Sisson 217 (open Wednesdays, 9-5:00)
- (315) 379-9845 (24-hour crisis hotline)
- Email: renewalhouse@verizon.net

And please: if you see something, say something. If you see that someone that you care about is struggling, please encourage them to seek help. If they are unwilling to do so, Care Enough to Call has guidelines on whom to contact. Everyone has the responsibility of creating a college climate of compassion.