Syllabus for CS 105, Programming Languages (Information Every Student Must Know)

Norman Ramsey

Fall 2022

Contents

[A Tufts CS education] means you can pick up most basic/simple programming languages super fast; CS 105 especially is so, so applicable.

— From the Class of 2022 Senior Survey

Reporting from the industry world, 105 and 160 remain the most foundational and applicable for my everyday work—I refer back to my memory of [105] often.

— Alumnus, Class of 2017

Introduction and Welcome

Welcome to CS 105! Here you will learn to write code from scratch, in a language you have never seen before, in a way that sails through code review.

CS 105 serves as the fourth course in our required programming sequence. In this sequence, it is the only 100-level course. It is the most technically deep, the most intellectually challenging, and the one in which you'll develop the most diverse and highest-level thinking skills. You will think hard, stretch beyond the skills you have grown comfortable with, and emerge with a permanently higher level of skill.

CS 105 introduces you—through extensive practice—to ideas and techniques that are found everywhere in today's programming languages. These ideas and techniques infuse many languages you have not yet seen, and they will continue to infuse the programming languages of the future. Foremost among these ideas are functions, types, and objects. CS 105 also introduces you to the mathematical foundations needed to talk precisely about languages and about programs: abstract syntax, formal semantics, type systems, and lambda calculus. Mathematics helps you recognize ideas and techniques that you have seen before, even when they are disguised.

To provide a sane and sensible context in which to learn, CS 105 immerses you in case studies, conducted using tiny languages that are designed to help you learn. In any given case study, you may act as a practitioner (by writing code in a language), as an implementor (by working on an interpreter for the language), as a designer (by inventing semantics for a related language), or as a scholar (by proving mathematical properties of the language).

CS 105 develops high-level, flexible programming skills that you can transfer. For example, you will find you can apply your skills to projects written in older languages such as Perl or Java, in currently popular languages such as JavaScript, Python, Ruby, Scala, and Swift, and in weird languages that might be important in the future, such as Haskell, Rust, Agda, Coq, and who knows what else. No matter what language you work in, when you finish 105, you'll be writing more powerful programs using less code. In a profession where complexity is the enemy, your 105 skills will equip you to tackle the most demanding jobs—and succeed.

What will the experience be like?

CS 105 is as difficult as any course in our program, and it's a difficulty of a *kind* you are not used to encountering. I want every student to succeed, and you are more likely to succeed if you know what to expect.

How does 105 compare with 40?

For most students, CS 105 is as difficult as CS 40. Some students find 105 a little easier, and some find 105 a little more difficult.

If you know 40, you know that your feeling of accomplishment comes primarily from building the projects. The 40 projects are easy to talk about: they look big, and many of them sound impressive, like "I built an image compressor." Most important, the 40 projects require that you apply the same skills that you learned in 11 and 15, just to a greater degree. In brief, 40 yields to force.

CS 105 also has a couple of projects that look big and sound impressive, like "I built a type inferencer." But overall, 105 is much more about finding a small, elegant, precise solution to a small problem—usually a programming problem, but sometimes a math problem. As a result, compared with CS 40,

- CS 105 presents more intellectual challenges.
- CS 105 gives you many more opportunities to make a satisfying breakthrough, often by an "aha moment."
- In CS 105, you spend much more time feeling lost.

But even though 105 is quite different from 40, there are lots of skills that transfer. If you have taken CS 40, then

- You know how to read a long assignment and how to work away at it gradually.
- You have practice building modules with specifications, and you have practice testing against specifications.
- You have some experience translating some challenging concepts into code.

But you must also be aware that some of not all of your habits in CS 40 will necessarily transfer to CS 105:

- CS 40 may fool you into believing that every hour spent at the keyboard represents progress toward your goals. In 105, it is all too easy to spend hours going nowhere. You *must* learn to pause, step back, and think about where you are going.
- In CS 40, you can succeed without having much insight into algorithms, linked data structures (think lists and trees), or recursion. In CS 105, you will need insight into all of those things.
- CS 40 may fool you into believing that you can complete any difficult course if you just grind away doggedly. In 40, this approach works: it may not be much fun, but you'll

finish. And grinding away constantly has another benefit: when you're grinding, you don't feel bad and you don't feel lost.

In 105, this approach works badly or not at all. If you are always grinding, you will never have that quiet moment that gives you the *insight* you need to solve a problem. You can't brute-force 105. Instead, you should deliberately break up your work sessions with *frequent pauses*. Such pauses will help you a lot. Stop, get up and move, and let problems percolate in your brain. You will probably feel lost or uncomfortable for a short time, but in the longer run, you'll finish faster, and then you'll feel great.

Why do I feel lost?

To succeed in 105, you must understand that it is *normal* to feel lost. There is nothing wrong with you, and there is nothing wrong with the course—feeling lost is a consequence of the situation: where you're coming from and where you're going.

Where you're coming from is most likely 11, 15, and possibly 40. All of these courses use a single programming paradigm, and they use almost the same language: imperative programming with unsafe pointers and explicit memory management, close to the machine. Even the syntax and the data are the same—compared to what you can see in the world, and to what you are introduced to in 105, differences between C and C++ barely matter. If you've completed 40, you're going to feel like an expert. And rightly so! You *are* an expert. But you're a *narrow* expert, and 105 introduces you to the demands of a wider world.

The most important way in which 11, 15, and 40 are narrow is that their programming model is a thin veneer over the machine model. These courses don't rely heavily on functions or types, and they do not commit deeply to objects. (While C++ makes it possible to commit strongly to types and to objects, it imposes such staggering complexity that these techniques cannot be taught in introductory courses.) Your inexperience with functions, types, and objects makes you a beginner again. And beginners often feel lost.

The other factor that comes into play is that when you learn challenging new material, you feel bad. Here's the neuroscience:[1]

> Many labs have observed that these critical brain regions increase in activity when people perform difficult tasks, whether the effort is physical or mental… [The road is difficult], though, because these brain regions have another intriguing property: When they increase in activity, you tend to feel pretty bad—tired, stymied, frustrated. Think about the last time you grappled with a math problem or pushed yourself to your physical limits. Hard work makes you feel bad in the moment.

[1]Lisa Feldman Barrett, "How to Become a 'Superager' ", *New York Times*, Dec. 31, 2016

What should I do about it?

When a successful student suddenly feels lost and confused, it's discouraging. To keep up your courage, I recommend two steps:

- First, recognize that unpleasant feelings are a normal part of a learning process. Our job as professors is to set you problems that are more challenging than you think you can handle… and to support you as learn you can handle them. This learning process requires hard work, attention to detail, and a leap of faith. But if you persist, your unpleasant feelings of confusion will be replaced by satisfying feelings of mastery.
- Second, ask questions. "Is it normal that I'm confused here?" "Are you confused here?" "If I'm confused but I still want to make progress, what skill do I need, or what technique should I practice?" "If you're not confused, what skill are you using? What technique are you applying?"

Both these steps will help you feel better.

You will also need to learn to proceed with your work *despite* feelings of confusion, uncertainty, or anxiety. To help with that, you need support.

What tactics do and don't work?

In 105, the slow approach is the fastest one. Effective 105 students stop, think, pause frequently, and work in short sessions. These tactics can be hard to put into practice, especially if you have used other tactics to succeed in other courses. But the work tactics often preferred by beginners do not promote success in 105. I paraphrase Garth Flint:

> [Talented students] are very resistant to [a systematic, thoughtful, slow] approach. They do not want to plan; they want to do trial and error at the keyboard. They have not learned an important axiom of programming: "three hours of trial-and-error coding will save fifteen minutes of planning." (I wish I knew who came up with that. They deserve an award.) [Students] want to hammer keys [in Cummings for hours on end and] then complain [that the course is too much work].

If you approach 105 by hammering away at the keyboard, *be aware how the approach is working for you.* If the course starts taking too much time, you can adjust your tactics—but only if you are mindful of them.

If you have already learned systematic tactics for program design and implementation, you should be able to apply them to CS 105. If you have not had to learn a systematic approach—and many students have not—you will easily master a nine-step design process described in *Seven Lessons in Program Design*.

How does the class support my success?

To help you navigate the learning process successfully, we have built a variety of support structures into CS 105.

- Early in the course, *homework* is structured as a large collection of small problems. When the going is most difficult, this structure makes your progress visible, even when progress is incremental.

- Every homework is accompanied by *comprehension questions* (CQs) about the reading. These questions guide you toward reading those parts of the book that are most valuable for completing the homework. The comprehension questions focus your attention on the most relevant parts of the book, so we recommend you do them *first*.

  Comprehension questions are short, and if you understand the reading, you should be able to answer all the questions in just a few minutes. To achieve that understanding, however, substantial reading is usually required.

- Weekly *recitations* give you supervised practice working on problems that resemble the homework problems. Most recitations are scheduled when a new homework is just starting, so you can get off early, on the right foot. Recitations are small, which makes them an ideal place to ask questions. (If you have a question, others in your recitation probably have a similar question.) *Recitations are mandatory,* and they count toward your course grade.

- Face-to-face interactions are invaluable. CS 105 provides extensive *office hours*, conducted primarily by students who know the course and who know how to succeed. We spend a significant fraction of the course budget providing face-to-face office hours, so take advantage of this resource!

  Every student is expected to go to the instructor's office hours at least once, for at least five minutes.

- After every homework, we distribute *model solutions*. These solutions model the best work of an expert in the field.

- This *syllabus* contains a heavy helping of advice.

- On the first day of class, we provide tips from the instructor and advice from successful students.

How can I use office hours effectively?

CS 105 office hours take place in the Cummings building. Our help area is on the main hallway of the third floor, just before you get to the kitchen. This is a new space for us, but we are hoping to have a whiteboard where you will be able to write your name in a column that matches your question or topic.

When you want help, go to the help area. Add your name *and your topic or question* to the queue written on the board. Your TAs have been instructed to prioritize *groups* of students who want to talk about similar issues, or who have similar questions. These groups are called *affinity groups*. You can also get help as an individual, but you may have to wait longer than when you are part of an affinity group. (The TAs on duty will help you form affinity groups.)

Office hours can get crowded, and not all students thrive in crowds. You are always justified in telling a TA, "I would like to go somewhere quiet to talk about this." Your TAs have been instructed to ask. (They have also been instructed to ask, politely, if other students can join the quiet discussion.)

What can I expect from teaching assistants?

The TAs are not there to debug your homework. They are there to help you learn. And the TAs who provide the best help and who help students learn the most are the ones who guide students with questions.

The questions that TAs ask are meant to be timeless and to apply to many problems. Eventually, you will be able to use the questions yourself, to help your own progress. Questions range from the technical ("what algebraic laws apply to this function"?) to the conceptual ("in your own words, what is meant by $\Gamma \vdash e : \tau$?") to the procedural ("how much of your 105 time is spent working alone?"). Expect questions about your work, your understanding, and your study practices.

Here are some other things you can and cannot expect from TAs:

- TAs will conduct themselves professionally—this is their job. You can expect them to treat you with professionalism and respect.

- TAs will be committed not just to your material success but to your well-being—I have told them that your well-being is our top priority.

- When it's your turn to be helped, TAs will always be willing to help you in a quiet place, or in private. I hope they will remember to ask you, but if not, a TA will always agree to help you in private.

- TAs will ask you a lot of questions. Questions will be framed with good will and with the intention of getting you on track to succeed. Many questions have been designed before the course starts, for use by the entire staff. When you need help with code, expect questions based on our nine-step design process.

- TAs will tell you when they don't know or aren't sure. They can't know everything, but they know what they don't know.

- TAs will push you to interact with other students who are working on similar problems, trying to understand similar topics, or confused by similar issues.

- TAs will push you to take responsibility for your own learning. You can expect them to want to understand what you are doing to study, what you are doing to prepare, and what you are doing on the homework problems themselves. You can expect TAs to ask about your study practices and to help you improve them.

- TAs will prioritize affinity groups over individual help.

- TAs will not jump you to the head of a queue just because your question is short. But they might jump you to the head of a queue if you are part of an affinity group. If you think you need help fast, form an affinity group.

What if I need special support?

Tufts University values the diversity of our students, staff, and faculty; Tufts recognizes the important contribution each student makes to our unique community. Tufts is committed to providing equal access and support to all qualified students through the provision of reasonable accommodations so that each student may fully participate in the Tufts experience. If you have a disability that requires reasonable accommodations, please contact the StAAR Center at `Accessibility@tufts.edu` or call 617-627-4539 to make an appointment with a representative who will help determine appropriate accommodations.

Accommodations cannot be enacted retroactively; if you need an accommodation, you must ask for it in advance.

How must I support my own success?

Everyone on the course staff is committed to your success. But as we do our part, you must do your part. Take advantage of the resources we offer. (They're described in this syllabus.) Most important, *do not allow yourself to become socially, emotionally, or intellectually isolated.* To prevent isolation, I recommend that you join a study group, and that when you can, you work in Cummings. The informal support network in our physical space has historically been strong; participating in this culture is one of the unique advantages of getting your computer-science education at Tufts. Don't miss out.

As you will see throughout this syllabus, there is more you can and should do:

- Go to lecture. Using a pen, pencil, or stylus, take notes.
- Go to recitation. Read the homework first.
- Don't try to swallow the textbook. To guide your reading, use the comprehension questions and the reading recommendations on the homework.
- Look at the homework problems as soon as they come out. Think about them a little bit every day.
- Talk about the homework problems with your friends and classmates. Some classmates may become new friends.
- Join a study group.
- Don't work alone. When you can, work in Cummings or in a study group.
- Take advantage of office hours.
- If the going gets tough, have another look at this syllabus.
- *Read* the handouts that describe experiences of top students in past classes and the instructor's tips for students.
- Build a "105-free zone" into your weekly schedule. Honor the zone even if you get behind, so that at least once a week you are sure to get a break from 105. (Because a homework is due every week, there is no weekly break built into the course.)
- Again, finally, and most important, do not allow yourself to become socially, emotionally, or intellectually isolated.

105 is a 100-level course. What does that mean?

Unlike 11, 15, and 40, CS 105 is numbered above 100. Formally, this means only that unlike 11, 15, and 40, 105 counts for graduate credit. But informally, a 100-level course carries additional expectations:

- *It's not just code; it's math.* All required 100-level CS courses have both code and math; the combination of code and math is the essence of the field. 105 is mostly code, but it still has lots of math; 160 is less code and more math; and 170 is almost all math.
- *You are responsible for your own learning.* You can't learn everything you need to know just by showing up at lecture. Lecture will give you the keys to the hard parts, but you need to tackle the book, and you need to fill in the gaps. Use the resources we provide you.
- *Junior standing is recommended.* 105 requires a degree of maturity that is not usually expected of freshmen and sophomores. Many students do take 105 as second-semester sophomores, but unless they are also strong in math, these students often struggle. It is reasonable to postpone 105 until your junior year.

What logistics do I need to know?

All the course information is on the home page at https://www.cs.tufts.edu/cs/105/. The splash page contains a guide that show what you need to do each week.

Announcements are posted to *Piazza*. Piazza is also, as usual, a forum for questions and answers.

What will we learn?

CS 105 is built around two sets of skills:

- *Programming skills* that exploit the best of the best programming-language features
- *Mathematical reasoning* about code

The programming skills contribute to your professional practice, getting you ready to code from scratch in a language you've never seen before. The math contributes to several outcomes:

- Math helps you communicate clearly about languages, language design, and language features.

- Math is a way of seeing patterns in the world. When you're confronted with a language you've never seen before, your mathematical experience will help you know what questions to ask, and it will help you recognize and identify the elements that you *have* seen before.

- Math is an ideal way to specify what programs do: it's clearer and more precise than informal English, and it's cleaner and more streamlined than a reference implementation.

- Experience with programming-language math will help you evaluate future claims about languages (for example, claims about security).

Here are some of the detailed skills you will develop:

- Read and write *precise specifications* of how languages work

- Understand how it is possible to *prove universal truths* that apply to *any* program written in a given language

- Design code using *algebraic laws*

- Write and reason about *recursive functions*

- Capture common patterns of recursion in *higher-order functions*

- Recognize and exploit common higher-order *list-processing functions*

- Program with first-class *functions as data*

- *Prove correctness* of code-improving transformations

- Express rich control structures using functions as *continuations*

- Design and implement *polymorphic* functions, methods, and data structures

- Understand the merits of *polymorphism* in programming

- Use *types* to guide the construction of code

- Understand in detail what are the merits of *type checking* and how type checking works, including polymorphic type checking

- Understand in detail what are the merits of *type inference* and how type inference works, including polymorphic type inference

- Describe computations using the *lambda calculus*

- Hide information using *abstract data types*, *modules*, and *interfaces*

- Hide information using *objects* and *protocols*

- Reuse code using *inheritance*

What topics will we study?

You will develop the skills above by studying these topics:

I. Functions
   - Recursion revisited: design by algebraic laws
   - Abstract syntax & (big-step) operational semantics
   - First-class, higher-order functions
   - Functions as continuations

II. Types
   - Core ML
   - Typing rules (monomorphic and polymorphic)
   - Type checking
   - Type inference

III. Theory
   - Lambda calculus & small-step operational semantics
   - Recursion and fixed-point operators

IV. Data abstraction
   - Abstract data types, modules, and interfaces
   - Objects, classes, and protocols
   - Inheritance and the design of class hierarchies

How heavy is the workload?

The workload in 105 is heavy; it counts for 5 semester hours of credit. Expect a demanding homework assignment almost every week. Most students say 105 demands as much time and effort as 40.

What does the workload consist of?

The work is mostly programming assignments. These assignments are significantly more challenging than the assignments in CS 15 and CS 40, but most of them are also much smaller: many solutions take 10 to 40 lines of code. Many of the assignments use software that comes with the text by Ramsey.

The homework also has a significant theory component. You will prove theorems using existing theory, and you will also develop new theory of your own, which will describe how a language feature might work. Early assignments are either "mostly programming" or "mostly theory;" later assignments mix programming and theory; and some assignments ask you to apply theory to write code.

The course will have no exams.

How does my work affect my grades?

Your course grade is based on *my judgment of the quality of your work* and the degree of mastery you demonstrate. My judgment is influenced by your written work, by your class participation, and by your examination scores, but I give heavy consideration to written work, as indicated by this approximate system of weights:

Reading-comprehension questions (with each homework)	10%
Homework exercises	70%
Recitations	10%
Office-hour visit to the course instructor	5%
Participation in other class activities	5%

The weights may be adjusted at my discretion.

How am I graded on reading-comprehension questions (CQs)?

A set of reading-comprehension questions is associated with each homework assignment. Each such question guides you to a specific part of the reading and lets you know which homework problems you are ready to tackle. Each comprehension question is graded as completely correct, mostly correct, or incorrect. When all questions are mostly correct, a homework earns Very Good grades for reading comprehension. Excellent grades may sometimes be earned for completely correct answers, and a majority of mostly correct answers earn Good grades. Reasonable attempts earn Fair grades, and substantially incomplete work earns Poor grades.

How am I graded in recitation?

CS 105 requires you to learn many new ideas and techniques. Before you can make progress on the homework, you need to understand the ideas and techniques. Recitation helps; each week it gives you practice working relevant exercises. Each recitation is graded on a three-point scale:

- **Very Good** recitation participation means *you contributed* a question, answer, idea, or solution, and also *engaged your classmates* to help them contribute. Your recitation leader will model for you what this means. **Very Good** recitation participation also requires that you come to recitation having looked at the homework and that you are prepared to talk about the reading-comprehension questions.

- **Good** recitation participation means you contributed, but you did not engage your classmates. Perhaps you spoke too much and didn't leave room for others to contribute, or perhaps you simply behaved as if others weren't there. Perhaps you had little idea of what was actually on the homework or you couldn't say anything about the reading.

- **Fair** recitation participation means only that you showed up.

We drop the ~~two~~ three lowest recitation grades, so if you have some bad days, it won't affect your final course grades. This policy also enables you to miss up to three recitations without penalty.

Although we drop the three lowest recitation grades, we still expect you to attend every recitation. *There is no such thing as an "excused absence" from recitation.* If you need to miss a recitation, it is kind to let your recitation leader know, but please do not ask them to officially excuse you. If you need to miss more than three recitations, explain the situation to your advising dean, and have your dean make contact with me.

How am I graded on my office-hour visit?

One-on-one meetings forge connections that are hard to make in any other way. To encourage such connections, I count up to five minutes of office-hour visits as part of your course grade. Each minute you spend in conversation with me during my office hours will earn one percent of your overall course grade, up to a possible total of five percent. To earn full credit, you must come to my office hours *by the end of October* (5:00pm on Monday, October 31). While you may find it helpful to talk about homework, class, engineering, or Tufts overall, any mutually agreeable topic of conversation is acceptable. Even the Red Sox.

Office-hour visits after the end of October still count toward your course grade, but not for the full five percent.

How is my participation graded?

A portion of your grade is based on your participation in the class. Participation demonstrates your commitment to actively managing your own learning. This commitment can be demonstrated in many ways:

- Asking appropriate questions in class
- Answering questions when called on in class
- Helping other 105 students in Cummings
- Organizing a study group and meeting with it at least twice
- Asking appropriate questions on Piazza
- Answering questions well on Piazza
- Submitting evidence that you wrote online end-of-term course evaluations

Nobody has to do *all* of these things; you can earn top grades for class participation by doing just a few things well. In particular, nobody is required to speak in class—but everybody should be prepared to answer questions if called upon. What questions are appropriate? Any question about programming languages. However, it may not be appropriate to insist that every question be tracked to its lair and answered. If a question becomes inappropriate during class, I will let you know.

If you organize a study group and it meets at least twice, *one* member of the study group should post a private question on Piazza (to "Instructors"). That question should name the participants and should identify two of the dates on which meetings took place. *One* private question results in participation credit for *all* persons named in the question.

How is my homework graded?

Your homework grades are based on the course staff's judgment of the quality of your work and your mastery of the material. Grades are assigned a coarse five-point scale:[2]

- **Excellent** work is outstanding in all respects. To be ranked Excellent, the work must truly excel; that is, it must exceed expectations in some way.[3]

  Excellent *documentation* addresses exactly the key issues, and degree of detail is exactly appropriate.

  Excellent *code* is very well thought out and implemented. Excellent code shows mastery of new language features and idioms. On the rare larger assignments, Excellent code shows evidence of thorough attention to abstraction and modularity. **Excellent code is so simple that it obviously has no faults.** Instructors will see no obvious ways to make excellent code shorter or simpler. Excellent code is laid out consistently and uses scarce vertical space well. Excellent code is of such high quality that the course staff would be happy to maintain it.

- **Very Good** work is of high quality in nearly all respects. An assignment that does everything asked for, and does it well, earns a grade of Very Good.

  Very Good *documentation* addresses most key issues, with a good amount of detail.

  Very Good *code* shows correct, idiomatic use of new language features. On the rare larger assignments, Very Good code shows that some attention has been paid to abstraction and modularity, although one or two opportunities may have been overlooked. Very Good code contains well-chosen names for functions and their parameters, so that it is easy to guess what functions do. Instructors may see one or two ways to make Very Good code shorter or simpler. Layout is consistent and uses scarce vertical space well. Small errors may be evident from reading the code.

- **Good** work demonstrates quality and significant learning.

  Good *documentation* covers some key issues, but significant issues may have been overlooked or may be covered with insufficient detail. Vague generalities may appear where precise specifics are expected.

  In Good *code*, individual functions are well organized and readable. Most names are well chosen, but there may be some exceptions. On the rare larger assignments, opportunities for abstraction and modularity probably have been overlooked. Good code gets the job done, but possibly in a way that could be shorter or simpler. Layout may be inconsistent in a few places. Errors may be evident from reading the code, but instructors will believe that code could be made correct with only modest changes.

- **Fair** work is lacking in one or more aspects; key issues need to be addressed. "Fair" is the lowest satisfactory grade.

  Fair *documentation* shows evidence of effort, but the degree of coverage and detail is significantly short of what the course staff believe is needed to foster success.

  Fair *code* contains significant faults. Instructors may not be able to figure out what all functions do. Layout may be inconsistent or waste scarce vertical space (e.g., every other line may be blank). Fair code may show evidence of a 'clone and modify' approach to program construction. Names may be poorly chosen. Possibly a Fair program could be replaced by code half its size. Given Fair code, instructors may believe major changes would be required to make the code correct, or instructors may be unable to understand *why* the code might be correct.

- **Poor** work shows little evidence of effort or has other serious deficiencies. "Poor" is an unsatisfactory grade.

  Poor *documentation* may fail to address key issues or may address them perfunctorily.

  Poor *code* may be undocumented or inappropriately documented (e.g., overcommented). Poor code is often lengthy, out of proportion to the problem being solved. Poor code may be laid out on the page in a way that is hard to read. Poor code often shows evidence of its history: extra copies of functions, unused logic left lying around, **old code commented out**, and so on. **Poor code may be so complex that it has no obvious faults**.

In addition to the grades above, some work might receive one of these other grades:

- **No Credit** is received for work not turned in, for parts that are incomplete, or for work that is non-functional or appears to bear no relation to the problems assigned.

  No Credit is received for work that is deemed to be plagiarized or that you cannot explain. Code submitted for CS 105 may be examined for potential plagiarism using automated heuristics. Plagiarism is a form of academic misconduct, which is unacceptable at Tufts. If academic misconduct is suspected, appropriate steps will be taken. A pattern of academic misconduct may lead to more severe penalties, including a failing grade for the course.

- **Not Sufficient** is received for extra-credit attempts that are not sufficient to earn credit. The difference between Not Sufficient and No Credit is that Not Sufficient shows up in your grade record, so we both know you made the attempt.

In a typical class, a consistent record of Very Good homework, together with commensurate examination grades, will lead to

[2] It is the same scale used by the National Science Foundation, by Consumer Reports, and by CS 40.

[3] By definition, it is not possible for the entire class to excel. The normal top grade is Very Good, and students who consistently produce Very Good work earn A's. Grades of Excellent are awarded only in cases of true distinction. This means, for example, that if everyone in the class turns in a perfect solution, all those perfect solutions are judged Very Good.

a course grade in the A range. If some work goes "above and beyond" and is rated Excellent, a grade of A+ is possible. Work rated Good corresponds to a wide range of passing grades centered roughly around B. Work rated Fair will lead to low but satisfactory course grades; if a significant fraction of your work is Poor, you can expect an unsatisfactory grade (D or F).

No Credit is a disaster. If 10% to 15% of your work is awarded No Credit, it is likely to cost you something like one full letter grade. Any grade, even Poor, is dramatically better then No Credit. Why? Because the worst thing you can do is to skip problems. Every problem you attempt will teach you something; when you skip a problem, you learn nothing.

I plan to give you estimates of your letter grade, but for the first time, I have eliminated all exams from the course. As a consequence, I cannot use historical data to project letter grades. After each homework assignment, I will give you an estimate, but once I have an accurate baseline, those estimates are likely to be adjusted.

What is a "minor deduction"?

Sometimes there is a fault in your code that needs to be corrected, but that does not warrant dragging a grade down from Very Good to Good or from Good to Fair. That sort of fault typically receives a "minor deduction." Here's the prototypical example:

> *It is never correct to write* `if P then true else false`. *Clean code always says just* `P`.

A minor deduction is the rough equivalent of a "point off" out of 100 points.

What if my homework is graded incorrectly?

If we make a mistake in grading your homework assignment, you have ***seven days*** after the return of the assignment to call the mistake to our attention. File a request using the form at https://www.cs.tufts.edu/cs/105/regrade. We will reassess your *entire* assignment and assign a new grade. The new grade may be higher or lower than the original grade.

What does a grade of D mean in the course?

The Tufts Bulletin states that a grade of D may be awarded in cases where a student submits work that is "unsatisfactory but allowable for credit." In CS 105, work that is allowable for credit earns satisfactory grades (Fair or above); work that is not satisfactory earns a failing grade (Poor). For this reason, D grades are not awarded: a record that is unsatisfactory overall earns an F.

Extraordinary circumstances may justify raising an F grade to a D. Such circumstances are very rare—roughly once every five years—and they apply at the sole discretion of the course instructor, whose decision is final.

How can I find my grades?

After the first homework has been graded, and throughout the semester, grades are available at https://www.cs.tufts.edu/comp/105/grades/. Supply your *departmental* userid and password. (These are the same credentials you use to log into the homework servers.)

How should I interact with people?

Engineering is not a solitary profession. To maximize your chances of success in 105 and beyond, I have designed some interactive experiences into the class.

- *Discussions with classmates*. Programming is a creative process. To help you think creatively, understand what the homework problems are asking, and discover paths to solutions, I encourage you to discuss the questions with friends and colleagues. You will do much better in the course, and at Tufts, if you find people with whom you regularly discuss problems.

  Once you start drafting code or algebraic laws, however, discussions are no longer appropriate. *Each program, unless explicitly assigned as a pair problem or a recitation problem, must be entirely your own work.* Your code, your algebraic laws, and the contracts for your helper functions must all be kept private.

- *Practice work with classmates*. Recitation will give you practice working out problems with the help of your classmates, under the supervision of a recitation leader. Your recitation leader will model and identify appropriate ways of talking about problems.

- *Informal interaction in Cummings.* When you are working on 105, we encourage you work in Cummings. You will be able to find other students who are also working on 105, and you may overhead some interesting conversations. You could get help that you didn't know you needed.

- *Deep work with one classmate*. Much of the programming in 105 is about your individual understanding of new language features and new ideas, and to develop this understanding, you will tackle most programming problems on your own. In the real world, however, substantial artifacts are seldom built by individuals working alone. CS 105 will therefore provide you with some opportunities to build something substantial by working deeply with one classmate, not just on the ideas, but on the code itself. This is *pair programming* (more at Wikipedia).

  In CS 105, pair programming is always an opportunity, never an obligation. You are not required to work in pairs, and typically about 20% of students choose to work alone. If you do choose to work in a pair—which I recommend—know that *no single pair may work together on more than three* *assignments*. If you need help finding a partner, advertise on Piazza.

- *Interaction with faculty*. One of your jobs as a student is to get to know some professors. Come to my office hours! Questions about the course, what we are studying, and why we are studying it are always welcome, as are broader questions about computer science, the future of the field, and your role in it. If the scheduled times don't work for you, post a request to Piazza with three times that do work.

  – *Interaction with faculty during lecture*. A particularly useful form of interaction is the question asked in class. Questions are always welcome; if you have a question, chances are other people in class have a similar question. Ask it! One question that is always legitimate is "why are we doing this?"

- *Interaction online*. At any time of day or night, you can post a comment or question to Piazza. *If your question reveals your code, your algebraic laws, or contracts for your helper functions, you* ***must*** *make it private.* Otherwise, questions on Piazza should be suitable for public consumption. If your question contains no code, algebraic laws, or contracts, please make it visible to everyone, so you get participation credit for asking it, and your classmates have a chance to help you by answering it. (By helping to answer your question, your classmates improve their own grades for class participation. Similarly, you can improve your grades by answering other people's questions—but *your answers must not contain code, laws, or contracts that you have developed for the assignment*. Do as you would be done by, and everyone wins.)

  I will use information in SIS to enroll most of you in Piazza. But if you are not enrolled, you can enroll yourself using the CS 105 signup page.

  We make every effort to answer posted questions in a timely fashion. But if you have posted a question to Piazza and have not gotten a timely response, many questions are appropriate to post to Stackoverflow, which can respond very quickly indeed. If you use Stackoverflow, please follow our guidelines, which are online.

  Questions that include code, laws, or contracts that you are working on (or that you have written) are OK for Piazza, but *they must be "private."*

  Please don't send questions directly to the course staff using their personal email addresses. If, in a rare case, you have an issue that is not appropriate for the whole course staff to see, please send email to my personal account.

- *Interaction with course staff in real life*. The course staff are here to enhance your learning; it is part of their job. Please interact with course staff using the same professional manners and standards you would use in any workplace. Please also recognize that CS 105 is only *part* of anyone's job: a member of the course staff may be present but not actually available to talk about 105. You can see who is on duty by looking at the duty roster.

- *Interaction on holidays*. We do not guarantee to hold office hours on university holidays. To know whether office hours will be held, consult Piazza.

How much interaction is too much?

Interactions and discussions with classmates must take place in human language, at a high level. You must not discuss code, algebraic laws, or contracts for functions you design yourself. You must not exchange human language, algebraic laws, pseudocode, or other information that is expressed at the level of code. If you start communicating in code or at the level of code, including algebraic laws, you're breaking the rules.

- While I encourage shared work at the whiteboard or in notebooks, if your shared work is so detailed or low-level that there is only one reasonable translation into code, you are collaborating too closely. (This is why conversations about laws are proscribed.)

- Unless you are working with another student as part of a programming pair, *it is not acceptable to permit that student to see any part of your program, including algebraic laws and contracts*, and *it is not acceptable to permit yourself to see any part of that other student's program, including algebraic laws and contracts*. In particular, you may not test or debug another student's code, nor may you have another student test or debug your code. (If you can't get code to work, consult a TA or the instructor.) Using another's code in any form *or writing code for use by another* violates the University's academic regulations.

- *Algebraic laws have the same status as code:* only you and your programming partner (and the course staff) may see your algebraic laws.

- *Contracts for helper functions have the same status as code:* only you and your programming partner (and the course staff) may see your contracts. For purposes of this rule, a "helper" function is any function that is not *named* in the assignment—contracts for named functions that are assigned by us may be discussed freely.

- *Do not, under any circumstances, post a public question to Piazza that contains any part of your code, algebraic laws, or contracts.* Such questions must be private.

Suspected violations will be reported to the University's Director of Community Standards, who will investigate, decide if a violation has taken place, and if so, recommend an appropriate penalty. Be careful! As described in the university's academic-integrity policy, the penalties for violation can be severe. A single bad decision made in a moment of weakness could lead to a permanent blot on your academic record.

The same standards apply to all homework assignments; work you submit under your name must be entirely your own work. *Always acknowledge those with whom you discuss problems!*

So can I post my code on Github?

Code in a Github repository is visible to other students. So *no, you may not post your code to a Github repository*—unless you are very careful to make the repository private. It is certainly convenient to use Github for backup and to share your work with potential employers. But it **must** be private. It is against course policy for you to put your homework in a public Github repository—and because a public Github repository can facilitate misbehavior by others, it is considered a violation of academic integrity.

How do pair-programming interactions work?

In pair programming, you work with a partner under the following constraints:

- When work is being done on the program, *both partners are (virtually) present at the computer*. One partner controls the keyboard; the other watches the screen. Both partners talk, and control of the keyboard should change hands occasionally. If you are not physically in the same space, you should have an open audio channel. If bandwidth permits, you should also have open video channel.

- *You submit a single program under both your names*. That work gets one grade, which you both receive.

While we strongly encourage both discussion and pair programming, we are also charged with guarding Tufts's standards of academic integrity. The following policies help ensure that these standards are upheld:

- If circumstances, such as scheduling difficulties, make it impossible for you to work as part of a pair, you may *ask the course staff for permission* to divide an assignment into parts and to do some parts as a member of a pair and other parts as an individual. Such parts must appear in different files, and **each file must be clearly identified as the work of an individual or the work of a pair**. Work done jointly by the pair should be submitted by *both* members of the pair. **Files containing joint work must be identical**. If you as an individual modify a file containing joint work, and you submit the modified file, that act will be considered a violation of academic integrity.

- **It is never acceptable** to divide an assignment into parts and have some parts done by one partner and other parts done by the other. Submitting work done by someone else as your own will be considered an **egregious violation** of academic integrity. **Submitting individual work as the product of pair programming** is also a violation of academic integrity.

What if pair programming doesn't work for me?

In CS 105, you are never required to pair program. About 20% of students routinely opt not to pair program.

If you try pair programming and find it is not working, or if your programming partner disappears in the middle of your project, proceed as follows:

- Notify your partner that you wish to discontinue the pairing. You needn't explain yourself; a simple wish to stop is reason enough.

- Submit the work done in partnership at that point, even if it is incomplete or broken. That work is "community property" and both students may submit it.

- If you wish, follow up with an additional submission of whatever you complete on your own.

Do I have a right to pair program?

Pair programming is a privilege, not a right. If you foul up and don't fix it, I may revoke your pair-programming privileges. Fouling up consists in any of the following *unacceptable behavior*:

- Repeatedly failing to keep appointments with your partner
- Lying to your partner about what you have done
- Other unprofessional treatment of your partner
- A pattern of violating academic integrity
- Other similarly egregious offenses

If I revoke your pair-programming privileges and you believe I have done so unfairly, you may appeal to the department chair.

What is expected of my homework?

In this class, *you will learn most of the material as you complete the homework assignments*. The importance of homework is reflected in the weight it is assigned. Most homework for this course involves short programming assignments. Many of them are based on the text by Ramsey. There are also some larger programming assignments. And there is some theory homework, involving more proving and less programming.

As in most classes, it helps to start the homework early. But in 105, *starting early seems to produce unusually good benefits*. Many students report that if they start early, even if they don't appear to make much progress, a solution will "come to them" while they are doing something else.

Another reason to start early is that *if you get stuck, early help is a lot better than late help*. 105 is a big course, and your difficulties could be overlooked until they get out of control. Keep an eye on yourself, and remember that a short conversation during office hours or recitation can save hours of aimless frustration.

If you complete and understand all the homework assignments, you are almost certain to do well in the course and earn a high grade. If you miss assignments or don't really understand the homework, it will be difficult for you to earn a satisfactory grade.

The homework is *relentlessly cumulative*; if you miss something important in an early assignment, you'll be handicapped in later assignments. You can't make up for early misunderstandings by cramming on later ones—slow and steady wins the race.

How should my homework be submitted?

Homework for each assignment is submitted using a course-specific *submit script*, with a name like `submit105-impcore`. The submit script checks to be sure you have files with the right names, and so on.[4] On most assignments, there are some additional checks:

- Code you submit must be accepted by the appropriate compiler or interpreter *without any warning or error messages*.
- Unit tests submitted as part of your solution must pass, and the entire file must load, including testing, in at most 250 CPU milliseconds. *If you have created long-running test cases, place them in a separate file*.

And the submit script may ask for other information:

- The submit script for the first assignment should ask you how we should pronounce your name. A CS 105 instructor might answer "kaeth-LEEN FIH-shur" or "NORE-muhn RAM-zee."

Before running a submit script, `cd` to the directory in which you have placed your solutions. If the submission script complains, fix the problems and resubmit. You may submit and resubmit the same assignment as many times as you like. I encourage you to *submit work early and often, even if it is incomplete*, so that you can be confident the submit script will accept your work.

How should my CQs be submitted?

Reading-comprehension questions for all units are submitted using a single submit script, called `submit105-cqs`. The script looks for all files matching `cqs-*.txt`, and it submits each one appropriately. Just download the CQs text file from the course web site, edit in your answers, and run `submit105-cqs`.

How do I get access to the submit scripts?

We provide a submission script for each assignment. To get access to those scripts, you need to execute

```
use comp105
```

to ensure that these scripts are on your execution path. This `use` command should also give you access to interpreters for `impcore`, `uscheme`, and so on.

[4]We hope one day to get `provide` to take care of these checks for us, but we're not there yet. Please don't submit homework using `provide`.

It is very convenient to put the `use` line in your `.cshrc` or `.profile` file, but to work around a misfeature in `use` you will need the line

```
use -q comp105
```

Without the `-q` option you may have difficulties with `scp`, `ssh`, `git`, VNC, or `rsync`.

A submission script is named `submit105-`*name*, so for example the submission script for the first assignment is called `submit105-impcore`.

How should my homework be formatted?

Whether it is digital or analog, your written work must bear your name, and it must be neat and well organized. Clear English expression is required; *grammar and spelling count*.

Every assignment should include a README file that describes the work. This description must

- Identify you by name
- Identify what aspects of the work have been correctly implemented and what have not.
- *Identify anyone with whom you have collaborated* or discussed the assignment.

How should my code be written?

Your code should conform to our coding-style guidelines, which are online. These guidelines will help your solution be understood and modified by others. Your code is graded on its conformance to these guidelines, on your *explanation* of what you are doing, and on your code's functional correctness.

Pay special attention to the *offside rule*: indentation should be consistent with syntactic structure.

You will notice that the code from the book does not conform to all of the guidelines in the handout—especially the guidelines on layout. That's because the book code is written as a literate program. If you wish to submit your own code as a literate program, you may. And you may choose whether you wish to work with the `bare` or the `commented` versions of the code in the book.

How will the structure of my code be evaluated?

The structure and organization of your code is evaluated according to our general coding rubric, which is online.

How will the correctness of my code be evaluated?

Submitted code is typically evaluated by automated testing. Our test scripts aim to explain why you got the grade you did, and if you made a mistake, what it was. On most assignments, you can expected grades to be assigned according to the following criteria:

- To earn a Very Good grade, code must show no faults. (In rare cases, we may assign Very Good grades to code with faults. In such cases, faultless code often earns Excellent grades.)
- To earn a Good grade, code must show faults only in particular corner cases that course staff will have identified as compatible with quality and significant learning.
- To earn a Fair grade, code may have faults, but it must have no grave faults. Grave faults that might disqualify code from earning a Fair grade include misuse of standard output, or such run-time errors such as using an undefined name or calling a function with the wrong number of arguments.
- To earn a Poor grade, code must show significant progress toward solving a problem, even if the solution is substantially incomplete.

In summary, Very Good code is expected to be faultless. Faulty code normally earns a Fair grade, but if the faults are very minor, it might earn a Good grade, and if the faults are grave, it might earn a Poor grade.

Code that writes to standard output or standard error, unless such output is called for in the problem specification, is at risk of earning No Credit. Use `print` and `println` at your peril!

Should I use LaTeX to write theory homework?

Not unless you already know how to do it. LaTeX and the `mathpartir` package do make it possible to typeset clear, legible inference rules, derivations, and proofs. That makes it easy for the course staff to read your work. But unless you already have experience using LaTeX to typeset mathematics, I recommend against using it. LaTeX is hard to learn, and it provides terrible error messages. In CS 105, you will be learning plenty of other power tools; learn LaTeX some other time.

If, however, you already know LaTeX, you may benefit by emulating our LaTeX source code for a simple proof system or Sam Guyer's LaTex source code for typesetting operational semantics.

Then how should theory homework be written?

Use pencil and paper. (Microsoft Word, Open Office, and that ilk are even worse choices than LaTeX—they aren't set up to handle even simple math, let alone inference rules.) Of course, you need to submit PDF. Here are two ways to get it:

- If you have a smartphone with a decent camera, there are many scanning apps. Wirecutter recommends Adobe Scan, which is free. We have also had good results with SwiftScan (formerly ScanPro, formerly ScanBot). And Google Drive can scan photographs to PDF.
- As a last resort, the big copier in the CS office has a document feeder and a Scan button. The scan is formulated as PDF and is sent to an email address you designate.

Any PDFs you submit will be emailed back to you automatically. Use the automated email to confirm that the PDF was correctly transferred to the homework server, that it opens, and that it displays your work as you expect.

Once I have my PDF, how do I submit it?

To submit a PDF, you must get a clean copy to the department servers. **This feat cannot reliably be accomplished using VS Code.** VS Code is known to corrupt PDF files. If you don't know another way to do it, follow the procedure outlined below.

Supposing you want to get a PDF file to your account on the servers, on the server you can create a suitable directory with this shell command:

```
mkdir -p cs105/homework/current-name-here
```

Then,

- If you are using MacOS, try

  ```
  scp theory.pdf username@homework.cs.tufts.edu:cs105/
  name-here
  ```

- If you are on Windows, install Putty, and a similar command (`scp`) should be made available to you. Or there may be a GUI way to do it.

Every time you submit a homework with PDF, that PDF should be emailed to you. Check the PDF to be sure it opens without errors. If there are errors opening it, we cannot grade it!

How may I use the library and the Internet?

You may look in the library (including the Internet, etc.) for ideas on how to solve homework problems, just as you may discuss problems with your classmates. But using the library is never required; everything you need to know can be found in lecture, in recitation, on the course web site, or in one of the books.

Some students rely heavily on the library. Although this behavior is permissible, I discourage it. I assign homework problems not because I want to know the answers, but because *doing* homework is the best way for *you* to learn. While library skills are important in our profession, the homework in this course is designed to develop other skills that are even more important. Remember, you will not have the library with you when you go on job interviews!

If you do use the library, the Internet, or other outside sources, *your homework must acknowledge the use of these sources*, even if you find little or nothing useful.

How may I use code I find on the Internet?

Any code that is posted to the official CS 105 web site or to Piazza is permissible for you to use. You **must not** download any other code from the Internet. You certainly may not submit it as part of a homework assignment.

May I use code from the book?

Absolutely! Any code from either of the books is fair game for you to use. Most of this code is provided in machine-readable form in `/comp/105/build-prove-compare`. Help yourself!

Several of the later homework assignments—especially around type systems—are feasible only with the support of code from the book.

How should I use Wikipedia?

Don't. *Wikipedia is a* terrible *source of information on programming languages*. Many of the entries are just plain wrong, and Wikipedia's rules make it nearly impossible for experts to correct bad articles. You don't yet have enough experience to identify bad information, so *don't use Wikipedia for 105.*

What does an extra-credit problem mean?

Most homework assignments will offer opportunities to earn extra credit. *I use extra credit to adjust final letter grades.* For example, if your grade average falls in the borderline between A- and B+, I will assign you the higher grade at my discretion if you have done significant extra-credit work. *I will also mention extra credit if I write you a letter of recommendation.* Extra credit is just that: extra. *You can earn an A or A+ without doing any extra credit.*

What if I can't get my homework in on time?

Homework is typically due at *11:59 PM* on a Tuesday. We will grant an automatic extension of fifteen minutes at no cost to you. If you plan on submitting your work at midnight, you will have fourteen minutes for last-minute changes.

We expect your homework to be submitted on time. But we recognize that the vicissitudes of college life may interfere. If you have difficulty, you have several options:

- For ordinary difficulties, each student is automatically issued *seven* "extension tokens." By expending an extension token, you get a 24-hour extension on *all* deadlines associated with a single assignment: all parts of the homework, plus the reading-comprehension questions. The extension happens automatically; when you turn in any piece of work more than fifteen minutes late, the course software charges you one extension token.

  Expenditure of extension tokens is governed by these rules:

  – *At most **one** extension token* may be expended on any single assignment.
  – *When you are out of tokens, late homework will no longer be accepted.* It will be returned ungraded, and you will receive No Credit for the work.

- If an illness affects your ability to complete homework on time, your first step is to report the illness using SIS. This step alerts us about your illness. We will then make suitable arrangements.

- If you experience extraordinary difficulties, such as bereavement, family emergencies, or similarly unpleasant events, please begin by making contact with your advising dean. Please take this step *before the assignment is due*. Ask your dean to drop me an email or give me a call, and we will make special arrangements that are suited to your circumstances. "Special arrangements" might include a longer extension, but more commonly, we will try to find a way for you to skip portions of the homework.

After the 24-hour extension deadline has passed, model solutions will be posted, and there will be no further extensions.

What if my partner is out of extension tokens?

If you are pair programming with another student, and you need to submit work a day late, *both* partners are charged an extension token. If your partner is out of extension tokens, your only choices are to submit on time or to get a "no-fault divorce" and submit separately. After the divorce, both students have equal rights to all common work, but only the student who still has tokens can submit late. At submission time, *both submissions must be identified as individual work* (list `nobody` as the programming partner). The true situation should be explained in the `README` file.

What if I can't finish all the problems?

If you can't finish every problem, act tactically to maximize your grade:

- *Start* every problem. Turn in, if you can, a partial solution, or at least an analysis. Incomplete work might still earn a grade of Poor, or possibly even Fair.

  If you turn in nothing for a problem, you get nothing for it. "No Credit" means *zero*. That's very bad for your grade. If you have to choose between two problems of equal weight, it is slightly better to get two Poor grades than to get one No Credit and one Very Good. You can't just scribble down anything and expect to receive a Poor grade—the course staff have to be able to find some substance. But if you document some progress, that's *far* better than No Credit.

- Make sure all submitted code compiles. If you can't get it to compile, comment it out and submit the comment.

105 is very cumulative, so if you can't finish every problem, you'll also want to act strategically to maximize the chances that you can do better on the next assignment. Your best strategic choice depends on the details of the assignment, so ask the course staff.

What is expected in lecture?

In lecture, I expect you to maximize your own learning and to eliminate distractions that might interfere with other students' learning. *Attendance at lecture is a privilege, not a right or an obligation.* If the course staff thinks your activities might be distracting other students, that privilege will be revoked.

Should I take notes? How?

To maximize your learning, I recommend that you *take notes, sketches, and diagrams by hand.* Paper is good, with a pen or pencil. If you have a large touch screen and can take notes with a stylus, that works, too. I recommend *against* using a keyboard with a standard laptop or word-processing software:

- The neuroscience is quite clear that note-taking with pen or pencil activates most of the brain. Note-taking using a keyboard activates a much smaller region.
- Word-processing software is terrible for note-taking. Your notes should be about highlights and connections: good notes connect more recent material with material from earlier in the lecture; good notes contain diagrams; good notes contain arrows and boxes. Good notes are highly non-linear. A word processor is designed to produced polished final documents, not to take notes. Use the superior tool: pencil and paper.

You never need to copy the instructor's notes or slides; notes and slides for every lecture are available on the course web page. Use your own notes to make connections and to highlight points that you find difficult or that you want to remember.

What distractions must I avoid?

During class, *please put your cell phone on vibrate*. If you must take a call, please leave the classroom and do not return until you have finished your call.

The ultimate distraction machine is the laptop computer. Please refrain from any computer-based activity that is visible from a nearby seat and that might distract another student. Such activities include, but are not limited to

- Visible images or videos
- Anything involving continual movement on the screen
- Anything involving sound
- Conversations by text message or chat
- Games

If you cannot resist such activities, put your computer away at the start of class.

What is expected in recitation?

Recitation helps deepen your knowledge of the course material, and it helps you learn to work in a way that promotes insight and defends you against overwork and exhaustion. Recitation is structured around *practice* on problems related to current homework assignments.

- Before your recitation, you are expected to have looked at the current homework assignment, and you are expected to be able to talk about the reading-comprehension questions.
- During recitation, you will discuss and tackle problems. You are expected to contribute, to engage your classmates, and to make sure your classmates have room to contribute.

Recitation is mandatory, and your participation is graded.

What logistics and policies affect my homework?

Here is a summary of relevant information that is distributed throughout this syllabus:

- Comprehension questions must be submitted using the submit script `submit105-cqs`.
- Homework must be submitted using a submit script.
- You can get *one* automatic 24-hour extension by expending one "extension token." You have a total of seven such tokens. On any given assignment, you can expend at most one token. One token gets you an extension on *both* CQs and homework.
- Code must compile or load with no errors or warnings, and it must do so within 250 CPU milliseconds.
- Except when you are pair programming, another student must never see your code or your algebraic laws.
- Any homework question can be posted publically to Piazza, *unless* that homework question contains code, an algebraic law, an inference rule, or another part of the solution. Any question that discloses your work must be posted privately.
- If needed, you have 7 days to request a regrade, which you do via a Web form located at https://www.cs.tufts.edu/cs/105/regrade.

I'm not required to take 105. Should I take it anyway?

CS 105 is required for Computer Science majors. If you're not a Computer Science major, find the paragraph that applies to you below.

If you're getting a *Computer Science minor*, or if you're just *interested in computing*, you should take 105 if you want an intense experience that will broaden and deepen your programming skills. (Unless you have a professional or recreational interest in mathematics or logic, some of the theory parts of the course may be less interesting to you.)

If you're a *postbac student* or a *master's student*, you also should take 105 if you want an intense experience that will broaden and deepen your programming skills. Many of our postbac students, in particular, report that 105 was a highlight of their certificate program. Postbac students and master's students do need to be aware that as part of Tufts's commitment to residential learning, the course is designed to serve students who can interact with course staff during the day.

If you're a *doctoral student* or *MS/PhD* student, the value of 105 depends on what you are trying to accomplish.

- If your research is going to depend on your ability to design, write, and evolve good software, take 105.
- If you're curious about programming languages as a potential area of research, consider taking 105. Just keep in mind that 105 is designed to be required of all undergraduate students. That makes it light on research and heavy on practice.
- If you know you're interested in programming languages and you want to study core programming-language topics at a graduate level, 105 is going to disappoint you—it's not operating at the right level. 105 is a *first* course in the field. As such, it just touches on the rudiments of theory, and it omits many other interesting topics, including logical relations, denotational semantics, garbage collection, and logic programming.
- If what you want is to meet the proficiency requirement for the PhD qualifying examination ("functional programming or object-oriented programming with inheritance"), 105 is one of several options. 105 is a good source for functional programming, but if you are going for object-oriented programming, you are probably better off with 180 (Software Engineering).

  Keep in mind that the proficiency requirement is minimal, and 105 presents a broad and deep view of programming languages that goes beyond minimal proficiency. If that's what you're looking for, great! But it's an intense experience that may take time away from your research.

What skills must I have already?

CS 105 is the final course in our required programming sequence, and it calls on a broad and deep array of skills you are expected to have developed in earlier courses.

What basic skills do I need?

You must grasp basic algorithms, data structures, and good programming practice.

What Unix skills do I need?

You must understand the basics of files, directories, creating and editing files, printing, compiling and loading programs, and using `make`. You will be much, much happier if you also can write a simple shell script (`sh`) and use Awk and `grep` effectively. You can learn about such things from Kernighan and Pike.

What file-transfer skills do I need?

If, as we recommend, you do your theory homework on paper, you need to know how to convert it to PDF and to get the PDF onto a departmental Linux machine.

- If you use the Adobe Scan app on a smartphone, it stores every scan you capture on Adobe Document Cloud, from which you can download it to the servers. Or if you use SwiftScan, you can get it to upload to a service like Google Drive.
- If you use some other means of creating the PDF, getting it to the servers is up to you. Various students report good results with Swift File Transfer from Android devices and `scp` (from the OpenSSH suite) from computers.
- If you use the copier in the CS office, you will be emailed a PDF attachment.

What theory skills do I need?

You must be comfortable with basic discrete mathematics. You must be able to prove theorems, *especially by induction*.

You must have taken Discrete Math (Math 61 or CS 61). If you have not, you must produce some other evidence that you can reason precisely about computational objects.

You must be able to *write an informal mathematical proof*. For example, you should be able to prove that a `sort` function returns the same set of elements that it was passed.

You must be comfortable using basic mathematical formulas with "forall" ($\forall$) and "exists" ($\exists$) quantifiers, i.e., the propositional and predicate calculi.

You must know basic set theory, e.g., the mathematical definition of functions as sets of ordered pairs.

You must be *comfortable reading and writing formal mathematical notation*, or at least be able to look at it without running screaming from the room.

What programming skills do I need?

You must have substantial programming experience. If you don't, you will have difficulty keeping up with the homework.

A few homework assignments require some proficiency in C. If you have a strong background in C++, some details will be different, but your background should be sufficient.

Your programming experience should include work with dynamically allocated data structures and with recursive functions. You must be *comfortable writing recursive functions* in the language of your choice, as well as *proving that such functions terminate*.

You must have implemented some of the basic data structures and algorithms used in computer science, like stacks, queues, lists, tables, search trees, standard sorts (quick, insertion, merge, heap), topological sort, and graph algorithms. These topics are well covered in CS 15 at Tufts. Prior exposure to exhaustive search (backtracking) will also be helpful.

What other programming skills might help?

It really helps to have some facility with systematic software design. Our recommended nine-step design process is a solid starting point, but if you have practice with additional design methods, you will prosper. Many of the most effective design methods are grounded in formal reasoning about programs, including the following intellectual tools:

- *Loop invariants and termination conditions* as they apply to imperative programs
- *Contracts* for functions, including *preconditions and postconditions*
- *Termination conditions* for recursive functions[5]
- *Representation invariants and abstraction functions* for abstract data types

You can brush up on this material by looking at the article by Bentley on the reading list. Chapter 4 of Liskov and Guttag has a nice tutorial on reasoning about data, which you will find helpful in several assignments.

What is the most important skill of all?

The most valuable skill you can have for CS 105 is the skill of managing yourself. To do well in 105 or in any other course that involves programming, develop these habits:

- Think carefully about a problem *before* you begin to write code.
- *When having difficulty* writing code, stand up, *walk away from your computer*, and think about the difficulty. Try scribbling on a blank sheet of paper, or if you are lucky enough to be near a whiteboard, enlist it as your ally.
- Never be satisfied with a "working program," but strive for *the simplest, clearest, most elegant implementation*.

If you have these habits, the other prerequisites are almost irrelevant. If you don't, you can expect difficulty no matter what other background you have.

[5]When writing recursive functions, try to develop your understanding of the deep connections between recursion and loops; the ideas of invariants and termination conditions apply to both.

What books do I need?

You absolutely, positively have to have the book *Programming Languages: Build, Prove, and Compare* by Norman Ramsey. Unfortunately, the hardcover will not be available until November2022, so you will need the spiral-bound, August 2020 edition—older editions will not do.

- To get a book, you may be able to find a used copy, and you can certainly buy one from the Computer Science department—you can pick it up from Cummings. When you buy the book, nobody profits: the intellectual property is donated; the copies are printed by the University printing office; and the department sells the book for a sum just sufficient to cover the cost of printing.
- If you want to buy a newly printed copy, we will post detailed information on Piazza.

You need the short booklet *Seven Lessons in Program Design*, which is online.

The book *Elements of ML Programming* by Jeff Ullman is very useful, but just for a few assignments. If need be, you can get by without it; we have assembled an extensive ML learning guide that uses entirely free resources. We recommend Ullman's book anyway, because we think that spending the extra money will save you enough time to be a good tradeoff. If you do buy the book, be sure you get the ML'97 edition. It is available in the University bookstore, but you can probably get it more cheaply elsewhere.

What computers can I use?

The class is set up to run on Red Hat Enterprise 64-bit Linux, as installed on the departmental `homework` server. For remote access use `homework.cs.tufts.edu`. The software from the book will be installed on these machines, but you can also grab the software and compile it on your own computer; try `git clone homework.cs.tufts.edu:/comp/105/build-prove-compare`.

If you need an account for CS machines, please send email to `staff@cs.tufts.edu`. I recommend that you ask for `bash` or `fish` as your login shell.

What software can smooth my path?

I recommend a wonderful program called `ledit`, which is extremely handy for interacting with our interpreters. Try typing, e.g., `ledit impcore`, and you will be able to get an interactive editing loop with the `impcore` interpreter. The `ledit` program is already installed on the departmental servers, and it can also be downloaded from INRIA and installed on your own machine.

I also recommend using a "programmer's editor" such as VS Code, `emacs`, or `vim`. A most valuable feature of such editors is the ability to jump directly to the source location of an error.

For `emacs` you will need to learn the `M-x compile` command. For `vim` you will need to learn the `:make` command, and you will probably need to `set makeprg`.

How about Emacs tricks?

If you do use `emacs`, you will find your code much easier to manage if you enable `paren-mode`, which you can do by placing these lines in your `.emacs` file:

```
(require 'paren)
(show-paren-mode)
(setq show-paren-style 'expression)
```

I also recommend using the `M-x customize-face` command with face `show-paren-match` to set the screen to bright yellow on parenthesis match.

You can use the Emacs TRAMP facility to edit files directly on the remote server, by using file names like `/ssh:nr@homework.cs.tufts.edu:/comp/105/www/syllabus.md`. This allows you to edit on your own local machine, but every time you save the file, it is automatically sent to the `homework` server, where you can compile or run it.

What may I do with solutions?

I provide model solutions to all homework questions. Studying solutions written by a skilled, experienced programmer can help you learn. But I provide model solutions for your private use *only*. Please do not share them with other students, and please make sure they do not find their way into public places, archives, and so on.

Copying solutions, whether from me or from another student, is a serious violation of academic integrity. *Providing* solutions to be copied is equally serious.

What if I get sick?

If you get sick (or just test positive) and you have to isolate, we will do our best to support you. For lectures, we can try to find the videos that were shot when the course was taught online. Office hours are more complicated. We do not have the budget to dedicate a TA on call for virtual office hours, and experience over the past year shows that we should not ask a single TA to cover both in-person an virtual hours—it just doesn't work. Instead we will do our best to support you on Piazza:

- When you post to Piazza, let us know you are isolating.
- If you would like a real-time consultation, let us know when you expect to be available.

We will do our best to set up a Zoom call within 12 to 36 hours.

What if the professor gets sick?

This past summer I got Covid, and I missed six weeks of work. If it happens again and I miss a week or two, most likely Professor Townsend or Professor Foster can give lectures. But if I'm out for a long period, we will fall back on video lectures. These are the lectures that I designed and Kathleen Fisher recorded; they are normally used only when the course is taught online. But if they are needed, I'll find a way to get them onto the course web site.

What if I am repeating the course?

If you are repeating the course, we expect you to abide by the following policies.

- There is no acceptable use of model solutions from prior semesters. If you have kept any model solutions, destroy them.
- If you have written your own solutions for past semesters, it is acceptable to consult them for ideas, and it is acceptable to submit parts verbatim. Such use must be acknowledged.
- In every homework, your README file must note what work is new, what work is based on work from a prior semester, and what work is submitted verbatim from a prior semester. Even if *nothing* is from a prior semester, your README file must disclose this information.

  (README files are anonymized, but PDFs are not. If you wish to preserve your privacy from graders, consider removing your name from any PDFs you submit.)
- In every homework, you must cite collaboration with every partner with whom you worked on the assignment in a past semester—even if none of that partner work survives.

What else do I need to know about academic integrity?

The course operates under the university's academic-integrity policy, which you must know. In addition, you must know that in CS 105, *seeing another student's code* is an integrity violation, and so is *allowing another student to see your code*. I am mandated to report even the suspicion of an integrity violation.

When a violation is suspected, the situation is investigated by the Director of Community standards or a person they designate. These people are the only ones empowered to find a student "responsible" or "not responsible" for a violation. However, if a student is found responsible, the grade penalty, if any, is set by the instructor.

Different investigators have different skills in reading computer code. I have dealt with cases in which any computer scientist would identify code as obviously copied, but the Community Standards investigators were not always confident enough to find a student responsible for copying. To respond to this situation, I have implemented the following policy: If I am convinced by a preponderance of the evidence that a student has copied code, and if during the investigation the student does not admit to having copied code, then if that student is found responsible for *any* integrity violation, no matter how petty, the grade penalty will be a failing grade for the course. (If a student does admit to having copied code, the typical grade penalty is a zero for both the homework and the comprehension questions associated with the copied work. When a violation is part of a larger pattern of academic misconduct, a harsher grade penalty may apply.)

Where else might I go to get help?

The StAAR Center (formerly the Academic Resource Center and Student Accessibility Services) offers a variety of resources to all students (both undergraduate and graduate) in the Schools of Arts and Sciences, and Engineering, the SMFA, and The Fletcher School; services are free to all enrolled students. Students may make an appointment to work on any writing-related project or assignment, attend subject tutoring in a variety of disciplines, or meet with an academic coach to hone fundamental academic skills like time management or overcoming procrastination. Students can make an appointment for any of these services by visiting go.tufts.edu/TutorFinder, or by visiting go.tufts.edu/StAARCenter.

What if I have a problem with a TA?

If you have concerns about anyone on the teaching staff, please raise the issue with a Teaching Fellow or the course instructor. If you don't feel comfortable approaching anyone associated with the course, please reach out to one of the department TA ombudspeople (at this writing, Mark Sheldon and Karen Edwards).

An ombudsperson helps mediate conflicts that arise involving TAs. Specifically, they will help with the following situations:

- A student having a problem with a TA
- A TA having a problem with a student
- A TA having a problem with another TA
- A TA having a problem with an instructor

If you are involved in or aware of such an issue, please contact a TA ombudsperson at ta-ombudsperson@cs.tufts.edu or ta-liaison@cs.tufts.edu.

I'm different. Am I really welcome here?

Yes! We strive to create a learning environment that welcomes students of *all* backgrounds. If you feel unwelcome for any reason, please let us know so we can work to make things better. Talk to anyone on the teaching staff, or if that feels uncomfortable, try your academic advisor, the department TA ombudsperson (who can be reached by emailing ta-ombudsperson@cs.tufts.edu), our department chair, or your dean.

CS 105 can be especially difficult for first-generation college students and for members of groups that are underrepresented in computing, who may not have the family or social support that helps them develop their skills in "how to be a college student." If you are a first-generation college student or a member of group that is underrepresented in computer science, I encourage you to have your "five minutes with the professor" visit early in the term, and we can talk about your support system.

Does the design process really work?

Judge for yourself. Piazza, Fall 2020:

> **I started coded without writing algebraic laws...**
>
> For a few of the homework problems, I decided to start coding before writing the algebraic laws and it was going well for most of them! But I was stuck on one function in particular. I just couldn't get it to work! I was at the point where I was going to give up knowing that it was failing a few tests. So I decided to comment out the tests that weren't working and write the algebraic laws and call it a day.
>
> But something amazing happened. When I was writing the algebraic laws, I had a complete breakthrough. By thinking about the different cases without thinking about the code, I somehow was able to see things from a different perspective. And once I finished writing them, I deleted my code and rewrote the function using these laws and it worked.
>
> Anyways, the experience was so surreal that I felt the need to post it to piazza.
>
> TL;DR: write your algebraic laws before you code!

Acknowledgments

For help with this syllabus I owe thanks to Kathleen Fisher, who gave feedback on the whole thing. I also owe thanks to Jared Chandler and Alex King, and for help with syllabi in general, Annie Soisson.