Syllabus: CIS 443 Programming Languages

Brian C. Ladd

Spring 2025

Catalog Description

> `Comparative study of programming languages. Functional, logic and object-oriented paradigms. Syntactic and semantic issues in language design. Prerequisites: CIS 203. Spring.`

Overview

Consider: What **is** a computer? That is, what is the minimal *thing* that you would consider a computer? A computer has three core components: input/output channels, a processor, and a memory. The memory contains *data*, encoded into **bits**, representing numbers, text, and even *instructions* on how to interpret the data. Data (and instructions) are encoded numerically and the *instruction set architecture* (ISA) of the processor is typically very simple; *encoded*, *machine-language* programs in memory are long and complex, typically too much to hold in a *human* mind.

Computer scientists address complexity through **layers of abstraction**. Rather than programming in *machine language*, humans do better working at a "higher" level: hexadecimal rather than binary expressions, assembly language mnemonics, imperative `C` (or `Fortran`, `Java`, `JavaScript`, *etc.*) programs, functional `Scheme` programs, or logic-based `Prolog` programs. The problems examined in this class are: How can a *digital*, *binary*, *general-purpose* computer **execute** a program expressed in a higher-level language? What *features* do different kinds of programming languages have and what *models of computation* do they represent? How is a programming language *designed* and *implemented*?

Sebesta, *Concepts of Programming Languages*, suggests that programming languages can be compared fruitfully in terms of readability, writability, and reliability; *reliability* raises deep questions: can language shape the solutions we can describe? [1] We will look at a couple of languages using these dimensions as well as the problem domain for which they were designed.

A program in a high-level language can be run on a computer by translating it into machine language. This can be done all at once by *compiling* it or one statement at a time can be *interpreted*. Implementing *semantics-preserving* translation applies regular and context-free languages from the theory of computation to the problem of language processing.

Calendar Considerations

- The last day to **add**: 2025-01-27
- The last day to **drop**: 2025-02-10
- The last day to withdraw: 2025-04-07
- The last day to select S/U: 2025-05-09
- **Spring Recess**: 2025-03-08 – 2025-03-16 (days w/o classes)
- **April Recess**: 2025-04-17 – 2025-04-20 (days w/o classes)
- **Last Day of Classes**: 2025-05-09
- **Final Exam**: 2025-05-15 10:15 – 12:15
- Additional Calendar Information: SUNY Potsdam Academic Calendar.

Student Learning Outcomes

Course Outcomes

`Programming Languages` Upon completing this course, students should be able to:

1. **Scheme**. *Implement* any standard CS1 program in the `Scheme` programming language: structuring control with selection, iteration, functions, recursion, and higher-order functions; structuring data as atoms, lists, and attribute lists. (Synthesize) **[Solve Problems]**

[1] See Orwell's *1984* and its "newspeak" for one answer. Or Fox "News".

2. **Interpreter**. *Construct* an interpreter (with tokenization, parsing, and execution) for a subset of the `Scheme` programming language supporting expression evaluation, built-in and user-defined functions, closures, and, if there is time, continuations. (Create) **[Multiple Layers]**
3. **Variables**. *Differentiate* between lexical and dynamic scoping when analyzing variables' lifetime and scope in various programming languages. (Analyze) **[Multiple Layers]**
4. **Parameters**. *Trace* and differentiate local and non-local variable resolution of various parameter passing mechanisms: by-value, by-value-return, by-reference, and by-name. (Analyze) **[Multiple Layers]**
5. **Functions**. *Contrast* the computational mechanisms created when various constraints on function calls are relaxed (*e.g.* immediate transfer of control, halting caller on call, closing context on callee return). (Analyze) **[Multiple Layers]**

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.] `Assembly Language` addresses PSLOs #2 and #3: Students graduating from the Computer Science department at SUNY Potsdam are expected to be able to

A. Apply logic and mathematical proof techniques to computing problems, including computability, formal languages, and complexity of algorithms.
B. Solve problems through analysis and implementation of tested programs that use data structures and algorithms.
C. Program the multiple layers (e.g. compiler, operating system, network, assembly language) between a high-level programming language and the underlying hardware.
D. Evaluate ethical outcomes of professional policies, practices, and products at societal, organizational, and personal scales.
E. Apply, alone and in teams, responsible software engineering methodology, tools, and practices.

Assessment

This is a 400-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 *programming languages*. You will also be graded on how well you can communicate your understanding in English (the language of instruction), formal logic notation (when applicable), and in *code* in various programming languages. 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*. You should set aside time **every day** or so to play with concepts from this class. That is necessary because learning requires multiple sessions engaging with material. You will also need time to read the textbook **outside** the classroom. Any *slides* are not enough. You will fail the course if you do not read the book.

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: they are edited to avoid showing incorrect information. This makes it look like a "real" programmer does not make mistakes. Note that the difference between novice and expert programmers is in the *types* of mistakes they make.

Editing, rewriting the messy, perhaps error-prone work that produced the program into a clear, easy to follow result, is the secret. **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 engaging with the material. They serve as a place to jump off of 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.

The limited amount of time for work is because the rest of the class keeps right on rolling along and there is a lot of material to cover. The longer it is from when you practiced something, the harder the assessment will be and the class will get 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

Programming languages permits reflection on what **programming** is. Building language processors is an important skill to have because any sufficiently successful program you build will need some way to configure it *without recompilation* or to read/write/audit complex data files and these tasks need some sort of language processing.

Note that learning in an **active** process on the part of the learner (that is you): read and engage with the text and end-of-chapter problems; write program after program after program in `Scheme`; talk about solving problems and writing programs with your classmates; explain how to solve problems/programs in class; reflect on programming.

The *theme* in the list of your learning activities: **generating** answers. Example solutions (in books or class) give students the wrong idea about solving problems and writing programs. Solutions to real problems are not written down in a clean quarter of a page with no crossing out and no false starts. Real programs are built iteratively, a bit at a time. Syntactic bugs, then semantic bugs, and finally logical bugs are removed through *testing*. Finally, and only if necessary, the solution can be *rewritten* to look like it was generated without any false starts.[2]

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 ~~hole~~ 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 programming: read good solutions and engage with real problems over and over to build up your intuition. This is a **skill** that grows with practice.

Grading

Category	**Percentage**
Reading	No Grade
Practice	
Participation	0.15
Group Work	
Programs	0.25
Interpreter Project	0.40
Exams	0.20

Percent	**Grade**
>= 93	4.0
>= 89	3.7
>= 85	3.3
>= 80	3.0
>= 77	2.7
>= 73	2.3
>= 70	2.0
>= 67	1.7
>= 64	1.3
>= 60	1.0

Participation The class meets during its scheduled meeting time. You are **expected** to attend every class meeting. You are expected to prepare for class by *reading* the textbook and working on the *programs*.
There may be individual/group worksheets done in class. They count as part of the *participation* grade and cannot be made up (you may work them for experience/practice but not for credit).
If they run long, you may take them home to finish for the next class meeting.
Pogramming Assignments Homework There will be worksheets and questions for homework. Much focus will be on `Scheme` but some will be on more general programming language problems.
Scheme Programs Programs in `Scheme`. Will be given across the semester. Must run in `Racket`'s **Simply Scheme** or **R5RS** dialect. Or in your interpreter.
Scheme Interpreter A multi-phase program. Must compile and run with *Gradle* and `Java 21` or `Racket` (as above).
Exams Two exams and a final (weighted double). All exams will be in-class. First exam will be on the first third of the material; there will be a somewhat cumulative second exam because of how material is interrelated. The final exam will be cumulative. Exams problems emphasize explaining *how* a problem is solved and interpreting *what* the answer means. This does not mean specific answers are never asked for, just that the focus is on explanation.

Grading Criteria

This is a 400-level course. You are transitioning from a young *student* to a young, professional *computer scientist*. Professional presentation of your work is part of how it will be evaluated:

[2]Include enough documentation to justify your design choices for future programmers using your code.

- Code must *compile*. If code, as submitted, fails to compile (or run in the interpreter), I will **stop grading:** the assignment will be marked as *Missing*.
  When you are told in the assigment that a given program must use `make` or `gradle`, it is expected that you will use that tool *correctly*. **Manage your time** if you need to learn to use the tool.
- Code in this course must be *professionally documented*: file, class, and function headers. All document the *intent* of the code, including any **invariants**, legal ranges for parameters, and explanations of any *tricky* code.
  If you use code from other sources, you **must** give appropriate credit. It is **academic dishonesty** to present another's work as your own. Code from generative AI is *not* your work. Expect to go to Student Conduct if you are dishonest.
- Code must *detect and report* errors as specified in the assignment. Appropriate error messages are up to you. Behavior as specified in the assignment or as appropriate if not specified.
- The code you turn in must be *tested*, the results must be *reported*, and you must explain how to *repeat the tests*. You must document, implement, and run tests for your code.
  If you cannot describe *how* to demonstrate the correctness of your solution, it is unlikely that you actually know that it is correct.
- Each programming assignment must include an *up-to-date* `README` file. Summarize the problem, your solution, how you knew what and how to test.
  Up-to-date means that if an assignment builds on another, you must update the `README` to reflect the new requirements. At a minimum, you have a new *solution* to a different *problem* and must have devised new *tests* to demonstrate meeting the new *requirements*.
  The `README` can be in `.md`, `.org`, or `.txt` formats. Notice, in particular, that these are all editable, plain-text formats (possibly with markup).

Textbook and Readings

`/home/student/Classes/443/notes/readings`, available on the lab computers, contains the Sebesta text, *Concepts of Programming Languages*, 11^{th} **Global Edition**. It also has a copy of Harvey's *Simply Scheme* which we will use to learn something about Scheme and programming.

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.

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 you 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.

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.

AI and You

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.

Introduction

The explosive growth of *large language models* (LLM) and other *generative artificial intelligence* (GenAI) technologies means that there are many opportunities to use these tools, for good or ill, in one's education. The growth of possibilities has far outstripped guidance for appropriate and ethical use of these tools. This policy is aimed at addressing this issue for WAYS 103 Algorithms of Oppression.

To examine the use of AI in American Culture, students will, necessarily, be exposed to sites offering the use of LLM. You cannot examine what GenAI does without learning how to use it.

Students and staff (users) are expected to abide by this policy for **all** material that is *used in the classroom* or *turned-in for grading*. Failure to follow these guidelines will be treated as **academic dishonesty** as defined at SUNY Potsdam.

Calculators and Collage

Why shouldn't a first-grader use a calculator to do their addition worksheet? Because they are doing the worksheet in order to **learn** what addition is. The calculator *short-circuits* their learning.

Further, what if the calculator, rather than always being correct, simply looks at all the answers ever given for `1596 + 7589` and reports one, randomly selected, as the answer? The answer, `9185`, is a **collage** of other (published(?)) answers. It may or may not be correct and the student has no experience with which to judge.

This same *short-circuiting* of learning and the user's *inability* to judge answer quality is massively amplified by GenAI.

Published(?)

The question mark is important: LLM's must use **exabytes** (*millions* of *terabytes*) of input for training. That certainly includes work under **copyright** and other restrictive licenses. Given the nature of LLM answers, generated by random walks through the word network on which it was trained, the output is influenced by works that may or may not have been ethically sourced. With most models now being trained by other models, this taint attaches to each new generation of GenAI as well.

Note further that queries are copied by the GenAI and are used to train future models. This means that using personally identifying information, confidential data, or copyrighted data in a query means that information will end up in future answers.

Acceptable Usage and Citation

1. GenAI tools can be used to generate **educational tools**, that are not turned in, without citation.
   Flash cards, outlines, reading lists, and even diagrams can be produced by GenAI; these are all acceptable as long as users do not claim authorship nor submit them.
2. GenAI tools can be used for **inspiration** and **must** be cited.
   Inspiration means GenAI results are used to spark creativity and are **not** quoted or used to direct overall structure of the work.

3. Some GenAI tools can *explain* and help **improve** user writing. Such use **must** be cited at the point GenAI suggestions are added to the work.
   To fit in this category of usage, users must have written a draft **without** GenAI tools /before/ asking. GenAI changes to the work should be noted as specifically as possible.
4. GenAI can **write entire documents** (essays, diagrams) from an assignment description. This use is **unacceptable** and **prohibited**, even with citation.
5. GenAI can be used to generate inappropriate images, audio, video, and text, to simulate an actual person, group, or corporation misleadingly and to demean individuals. Note that the behavior, inappropriate image, fraudulently stealing an identity, or demeaning another, is a violation of SUNY Potsdam policy. The use of GenAI does not somehow make it acceptable.

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.

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? Performing Arts Center 223. My office hours appear on the course schedule and printed outside my door.

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

If you are seeking programming assistance, you should make a copy of your current code available to me. Easiest way to do that is to commit your current work to `git` and push it to `cs-devel` and then shoot me an e-mail or Discord message about where it is.

Except in so far as I am obligated to report sexual harassment, 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 grade book 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.

Additional Information

[Much of this text from SUNY Potsdam Syllabus Guidelines]

Accomodative Services

If you are a student with a disability and wish to discuss reasonable accommodations for this course, contact me privately to discuss the specific modifications you wish to request. If you have not yet contacted Accommodative Services, located in Sisson Hall Room 137,lease do so in order to verify your disability and to coordinate your reasonable modifications. For more information, visit the Accommodative Services website.

Counseling Services

As a student at SUNY Potsdam, you may encounter various stressors that can affect your learning and academic performance. The College Counseling Center (CCC) offers a range of resources to support students in their mental health, social well-being, and overall academic experience. Students seeking to speak with a licensed mental health professional can access free services through the SUNY Potsdam Counseling Center. To connect with our services, please call 315-267-2330, email counseling@potsdam.edu, or visit us at Van Housen 131.

Disruptive Student Policy

In order to maintain a fair, just, and safe College community environment, students are subject to SUNY Potsdam College regulations and are expected to abide by the Potsdam Pledge, the SUNY Potsdam Academic Catalog, and the Student Community Rights and Responsibilities (the Student Code of Conduct). Anyone who disrupts the academic learning environment (e.g., classrooms, labs, office hours, online platforms, emails, or other virtual communications) will be asked to leave for the day. Disruptions may include, but are not limited to, interrupting/speaking over other people, excessive use of foul or abusive or derogatory language, and repeated getting up/leaving/returning behavior. Students who exhibit aggressive/violent behavior in or out of classrooms will be asked to immediately leave. Disruptive students may be subject to disciplinary action, potentially including administrative removal from the course.

Food Insecurity

The Prometheus Alumni Food Pantry is located in 119 Barrington Student Union. The hours of operation can be found on their Facebook page or Get Involved. An Initial Intake Form is required of all visitors. Additionally, if you are living off campus and in need of supplemental food assistance you may be eligible for SNAP (Supplemental Nutrition Assistance Program) benefits. To determine eligibility and to enroll, you can visit mybenefits.ny.gov or call Maximizing Independent Living Choices (MILC) at 315-764-9442 x405. The campus has a registered dietician to assist you; you can contact them through the Student Health Services student portal on BearPaws. If you have questions on any of these services or need further assistance, reach out to the Bear CARE Program.

Land Acknowledgment

We acknowledge with respect the Mohawk Nation, the Indigenous people on whose ancestral lands SUNY Potsdam now stands. We are reminded by our presence here that we have the duty to live in harmony with one another and with all of creation. We are deeply grateful to the families and communities who care for this beautiful place. Beginning with colonization and continuing for centuries the Haudenosaunee Peoples have been dispossessed of most of their ancestral lands by the actions of individuals and institutions. We acknowledge our responsibility to understand and respond to those actions and to commit to working together to honor our past and build our future with truth.

Success Statement

Success in this course depends heavily on your personal health and wellbeing. Recognize that stress is an expected part of the college experience, and it often can be compounded by unexpected setbacks or life changes outside the classroom. I strongly encourage you to reframe challenges as an unavoidable pathway to success. Reflect on your role in taking care of yourself throughout the term, before the demands of exams and projects reach their peak. Please feel free to reach out to me about any difficulty you may be having that may impact your performance in your courses or campus life as soon as it occurs and before it becomes too overwhelming. In addition, I strongly encourage you to contact the many other support services on campus that stand ready to assist you including the Counseling Center & the Case Manager, Student Health Services, Accommodative Services, and your academic advisor.

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.