Teaching Dossier

Alicia M. Grubb
Department of Computer Science, University of Toronto

December 7, 2017
Version 1.1

Contents

1 Introduction

I am deeply passionate about university life and the personal development it affords. I feel elated when my learners make connections between concepts and see the world anew. My first exposure to pedagogy and teaching "at scale" occurred in my undergraduate degree when I worked as a tutor. Seeing the explicit coordination efforts and structuring of the course ignited my interest in curriculum development. This early experience helped me contextualize my efforts, and I began to reflect on each experience. Since then, I have had extensive experience in teaching and curriculum development, as I describe below.

I am poised to teach software engineering courses, for example, requirements elicitation and specification, modeling, design, and project management. I am confident to teach foundational computer science (CS) courses, such as programming principles, logic, algorithms, and hardware. In addition to these, I am also in a position to teach critical thinking, technical writing, applied statistics, and the social impacts of computing. In the remainder of this document, I discuss my teaching philosophy and experience.

2 Teaching Philosophy

My teaching philosophy is a manifestation of my core values: ***authenticity, equity, contribution, collaboration, transparency, improvement, and evidence***.

I aim to create an **equitable** and inclusive classroom. I make my course content accessible by providing a variety of examples from different domains, and I try not to make assumptions about the background of my learners. I aim to ensure that non-native English speakers are not disadvantaged in rubric design. I believe everyone can **contribute**, and we can all learn something from the lived experiences of each other. I thus lead with humility and focus on outcome-based learning. I value **collaboration** and am grateful for the contributions of my learners and my critics. I openly seek feedback, and acknowledge the impact of my mentors on my success. I strive to be **transparent** in my teaching and expectations of my learners. I use clear wording in assignments and tests, and communicate clearly and directly with my colleagues. I am honest when I make a mistake, and I share my weaknesses with my learners. I believe excellence in teaching is constantly evolving, adapting, and **improving**. Just as I want my learners to develop their critical thinking skills, I think we as educators must question whether our interventions are effective. I am data-driven and use **evidence-based** design science techniques to develop and evaluate education initiatives. I recognize the limitations of measurement and value constructivist views of evidence. Finally, I believe it is imperative to be my **authentic** self as an educator, and to embrace my own history and what makes me unique. I teach what I am passionate about and connect with learners because education is transformative.

3 Teaching Experience

3.1 Instructor and Teaching Assistant

While a graduate student at the University of Toronto, I worked as a teaching assistant (TA) every year. I strategically chose a variety of courses across all years of the CS program, but also service courses for non-CS majors (see Tbl. 1 for the full list). Rather than discuss my history chronologically, I highlight experiences, grouped by activity.

Large Group Instruction (Lecture / Tutorial). Upon starting graduate school, I was very excited to get in front of students and teach tutorials, and started with *CSC258: Computer Organization.* I focused on working through examples with the participants and creating an interactive experience. On my TA evaluations in 2008 (see my Evidence of Effective Teaching), one student commented "Alicia is the best TA I have had; she is well-prepared, teaches effectively, and appears genuinely excited about the course material", but I still believed that my teaching evaluations could become stronger. I didn't receive any negative or constructive comments before the end of the term when one student commented "talks too fast => erases board too fast". This was a formative experience, where I learned that I needed to actively solicit feedback if I wanted to **improve**. In the next term, I focused on my board skills, pacing my explanations, and checking in with my learners. My evaluations in 2009 improved, and one student noted "goes through the examples and explains them slowly and clearly. The tutorials helped to make clear the material greatly".

Working as a tutorial and lab TA for *CSC104: The Why and How of Computing* also proved formative because it was my first in-depth CS teaching experience with non-CS majors. Suddenly, my tacit assumptions about the prior knowledge of my learners were invalid. I loved the challenge of having to come up with analogies for concepts and applications of CS. For example, why would an art history major need macros in spreadsheets? I now believe all learners can benefit from computational or algorithmic thinking as a method of problem solving. This experience prompted me to seek my first instructorship with non-CS majors.

I taught first year chemical engineers as an instructor, for two offerings of *APS106: Fundamentals of Computer Programming.* My goals were to be an effective educator, and to excite my learners about programming. I changed the way I presented materials to accommodate different note-taking styles, and made all of my lecture content available online before lecture. My notes were a continuous document, which, unlike the slides used in other sections, gave learners extra time to take notes. I used active learning techniques (e.g., Think, Pair, Share) and brought visual props or demonstrations into the classroom. I used cue cards six times per term to get quick feedback from the students, asking targeted questions about course content, and soliciting general feedback about the course. These cards allowed me to anonymously assess the needs and progress of my learners, but also to directly address concerns. In an attempt to increase engagement, I used examples from a variety of domains including chemical engineering.

After the first year (2012), my teaching evaluations for APS106 were similar to those I had received as a TA, and better than the average for other instructors in the department and faculty[1] (see my Evidence of Effective Teaching). In the second year (2013), I was able to focus on my teaching, and saw **evidence** of my improvement in my evaluations. One student later wrote "She is thorough and clear in her explanations and teaches with energy which keeps students awake and interested. She often brought up examples from chemical engineering to make the class more interesting and relevant to our program. Overall she was my favourite instructor in first year engineering and I always looked forward to attending her class".

One-on-one Instruction. I have experiences with one-on-one instruction through office hours and the help centre as well as mentorship and projects. In the help centre, I focused on assessing the state of the learners understanding, and the help I could provide. Most of the time, I helped them frame their problem and reviewed course content. Sometimes, they needed space to explore their ideas or someone to listen to them; other times, it was an **authentic** connection over a shared joy or struggle. All these moments were important for retaining learners.

I have had the pleasure of supervising five undergraduate researchers (2015–2017). Since each arrived with their own strengths and history, I tailored my instruction to their needs. With the goal of having a substantial research **contribution**, we focused on setting goals, managing each others' expectations, and openly discussing issues. These experiences were guided by the learner. I mentored some in professional development, others in improving their writing, and all in choosing whether graduate school was the right fit for them. At the end of each collaboration, they had the option to co-author an abstract or a workshop paper if they were interested (see my CV for the full list).

In addition to mentoring undergraduate researchers, I have mentored other course instructors interested in creating a more **equitable** classroom experience. For example, I reviewed their lectures and assignment examples that presumed that CS students had common interests (i.e., the geek culture). One instructor later wrote "Alicia helped

[1] Department of Mechanical and Industrial Engineering, Faculty of Engineering

me become a better educator by showing me how to reflect critically on my conduct and behaviour, especially in the face of the intersections of imbalanced power dynamics".

Table 1: Teaching Experience in the Department of Computer Science, University of Toronto

Course Instructor		
Term	Course	Course Coordinator
Winter 2013	APS106: Fundamentals of Computer Programming	Markus Bussmann
Winter 2012	APS106: Fundamentals of Computer Programming	J. Christopher Beck
Teaching Assistant		
Term	Course	Supervising Instructor
Fall 2015	CSC300: Computers and Society	Mathew Zaleski
Winter 2015	CSC300: Computers and Society	Mathew Zaleski
Fall 2014	CSC300: Computers and Society	Mathew Zaleski
Summer 2014	CSC108: Introduction to Computer Programming	Craig Hagerman
Winter 2014	CSC300: Computers and Society	Susan Elliott Sim
Fall 2012	CSC258: Computer Organization	Steve Engels
Winter 2012	PMU199: Climate Change - Software, Science and Society	Steve Easterbrook
Fall 2011	CSC209: Software Tools and Systems Programming	Alan Rosenthal
Winter 2011	CSC302: Engineering Large Software Systems	Jennifer Horkoff
Fall 2010	CSC104: The Why and How of Computing	Alan Rosenthal
Winter 2010	CSC104: The Why and How of Computing	Alan Rosenthal
Fall 2009	CSC490: Capstone Design Project	Gregory V. Wilson
Winter 2009	CSC258: Computer Organization	Alan Rosenthal
Fall 2008	CSC258: Computer Organization	Alan Rosenthal
Other Teaching Positions		
Term	Position	Supervisor
Fall 2017	First Year Help Centre TA	François Pitt
Winter 2017	First Year Help Centre TA	François Pitt
Fall 2015	TA Trainer	Michelle Craig
2015–2016	Writing Fellow (Lead Writing TA)	Andrea Williams
Winer 2015	Mentorship TA	Diane Horton
Winter 2015	TA Trainer	Michelle Craig
Fall 2014	TA Trainer	Michelle Craig
2014–2015	Writing Fellow (Lead Writing TA)	Andrea Williams

3.2 TA Training and Curriculum Development

Writing Instruction for TAs. As the Computer Science Writing Fellow (2014–2016), I was given a mandate to create a "culture of writing" by training Teaching Assistants on teaching and evaluating writing, and working with course instructors to create writing-specific content as part of a broader Writing Across the Curriculum (WAC) program. In workshops, I taught course-specific training modules (e.g., Giving Effective Feedback, Using Rubrics, Low Stakes Writing, Reverse Outlining, and Working with Language Learners) and held benchmarking/feedback meetings. Benchmarking is when TAs and instructors meet to discuss an assignment and its rubric, and mark sample assignments together. In feedback meetings, TAs and instructors swap previous assignments and give each other feedback on how to improve their comments for learners (i.e., feedback-on-feedback). I tailored subsequent workshops based on recommendations, and TA impressions[2] of my workshops improved from 3.95/5 in 2015 to 4.33/5 in 2016[3]. This experience reinforced the importance of peer learning and **collaboration**. The WAC coordinator was so impressed with my efficient reuse of past training materials from other disciplines, that she invited me to train the future writing fellows across the faculty on project planning and time management.

Department TA Training. I coordinated and led the TA Training for all incoming CS TAs in 2014 and 2015. In a 3-hour workshop, I covered active learning techniques, running tutorials/labs, getting/giving feedback, marking and plagiarism, and professionalism and harassment. 180 TAs attended my training sessions and my training materials are still being used. On a five point scale[3], TAs rated these workshops highly (4.22), as they did my instruction (4.53), and their learning (3.88). In their feedback, TAs consistently reported the workshop as beneficial, and were very enthusiastic about additional training opportunities. For example, one wrote, “Good learning experience. Answered some questions I had and some I didn’t know I had”, while another wrote, “Engaging & fun, yet also useful”.

Curriculum Development and Review. While I have participated in various curriculum initiatives, I will focus here on the one I spearheaded in 2014. After grading the written assignments and term papers for *CSC300: Computers and Society*, for which the quality varied dramatically, I recognized the potential to engage students in writing and critical thinking as part of CSC300. I focused on structured technical writing of arguments rather than sentence-level mechanics. I improved the syllabus and introduced learning objectives throughout the course. I scaffolded the assignments, such that each assignment was built on the previous one while not penalizing past performance. I also introduced benchmarking sessions, similar to those discussed above, to ensure learners received specific and targeted constructive feedback on assignments in a timely manner. In the first term, we only saw limited improvements in the assignment quality. In later offerings, we offered opportunities for learners to resubmit their work, encouraging them to consider our suggestions to strengthen their argument, which resulted in visible improvements. We also introduced a debate component, giving our learners practice presenting their arguments verbally. We introduced writing, public speaking and debate into the tutorial exercises to give learners structured practice. Thus far, around 900 learners have received formative feedback as a result of this effort, and I continue to receive positive reviews from the course instructors.

Curriculum Research. The Department of Computer Science at the University of Toronto has a culture of education-based research and measuring the impact of teaching interventions. As the Computer Science Writing Fellow, I examined how students perceive authoring course blogs in the context of a large and linguistically diverse introductory computer science course. Prior course evaluations indicated that many students were resistant to writing in general and saw writing as unrelated to programming. Over two years, I collected student and teaching assistant survey data (N=Pre/Post, Year 1=350/206, Year 2=766/308). I found that writing course blogs improved students’ confidence in their own ability to write, but did not improve the quality of their writing or understanding of course concepts. More research is required to investigate how to persuade students to value writing activities. My research into students’ engagement with writing connects directly with improving the curriculum of *CSC148: Introduction to Computer Science*. In connecting with other instructors, I have also learned from the other active initiatives (e.g., inverted classrooms), and these studies have influenced my research interests in improving CS education.

[2]Feedback form asked: “Please describe your overall impression of this workshop.”
[3]Scale: 1. poor, 2. fair, 3. good, 4. very good, 5. excellent.

4 Evidence of Professional Development

Professional development is important in staying current with pedagogical practises and generating new ideas. While professional development often happens informally by connecting with colleagues and reviewing the latests literature, formal workshops have played a valuable role in my improvement.

4.1 Writing Training Workshops

As a writing fellow (2014–2016), I participated in nine days of training over two years on writing instruction. I also attended the Writing Instruction for TAs Showcase. In training we discussed a wide variety of topics and had many breakout sessions. My favourite sessions were the following:

- Theories of Writing Development
- Working and Communicating with Faculty & TAs
- Writing to Learn: Low Stakes
- Keeping it Real (and Fresh): Training and Empowering New & Experienced TAs

I also presented two sessions, Assessments & Research Projects: Highlights & Samples, and Planning and Managing your Hours for Maximum Impact.

4.2 Course on Professional Instruction

I audited *APS1003: Professional Education and Instruction* in 2013. I attended lectures, completed writing reflections, quiz, and exams. The course covered many areas, including the following topics:

- Models and theory of adult learning
- Learning styles and situational factors
- Developing learning outcomes
- Assessment and teaching for transfer
- Design of learning experiences
- Instructional methods
- Feedback concepts and methods
- Basics of mentoring.

This course improved how I thought about my curriculum development efforts, and gave me several frameworks on which to structure my initiatives.

4.3 Other Workshops

The Teaching Assistants' Training Program offers a series of workshops for TAs to gain expertise. I attended the following workshops.

- *Active Learning in Math-based Environment*, 2011
- *Making Your Syllabus Work for You and Your Students*, 2012
- *Preparing Your Teaching Dossier*, 2012

In 2014 I participated in a workshop entitled *Developing Learning Outcomes* offered by the Faculty of Engineering's Teaching Methods and Resources Committee. The goal of this workshop was to help instructors define learning outcomes specifically for their labs and tutorials and connect them with curriculum goals and engineering accreditation requirements.

5 Evidence of Effective Teaching

While a graduate student at the University of Toronto (2008–2017), I worked on a variety of courses (see Tbl. 1 for the full list). Here I provide evidence of teaching effectiveness in the form of student evaluations.

5.1 Summary of Evaluation Data

The data provided in Tbl. 2 includes my evaluations as a teaching assistant (TA) and instructor. Overall, I found the quantitative evaluations helpful in benchmarking my progress as an educator. TA evaluations were only collected for six terms in my department[4], but I believe this data demonstrates my conscious effort to improve each term.

I received full evaluations when I taught *APS106: Fundamentals of Computer Programming*. In my first year as an instructor (2012), my teaching evaluations for APS106 were comparable to those I had received as a TA, and better than the average for other instructors in the department and faculty[5] (see Tbl. 2). In my second year (2013), I changed aspects of my teaching (described in detail with student feedback), and saw evidence of improvement. My evaluation response rate (a proxy for class attendance) also improved, and was 20% higher than the department/faculty average in 2013. My learners' enthusiasm was unchanged in 2012; however, in 2013 my learners' enthusiasm increased (by 0.48) while the department/faculty average went down in both years (see Tbl. 2).

Table 2: Teaching Evaluations. APS106 evaluations also give department (*Dep.*) and faculty (*Fac.*) averages[5]. Scale: 1. extremely poor/inadequate, 2. very poor, 3. poor, 4, adequate, 5. good, 6. very good, 7. outstanding.

	Teaching Assistant[4]						Instructor[5]	
	CSC258		CSC104	CSC302	CSC209	CSC300	APS106	
	Fall 2008	Winter 2009	Fall 2010	Winter 2011	Fall 2011	Fall 2015	Winter 2012	Winter 2013
							Ins. (*Dep.*|*Fac.*)	Ins. (*Dep.*|*Fac.*)
Num. Responses	15	22	17	2	5	7	49	66
Overall	4.60	5.86	6.41	5.50	6.40	6.29	5.84 (5.18|5.26)	5.95 (5.28|5.38)
Presentation	4.27	5.59	6.18	5.50	5.60	6.71	5.31 (4.85|4.94)	5.60 (4.96|5.04)
Clarity	4.73	5.91	6.06	5.50	6.20	6.57	5.27 (4.88|4.98)	5.50 (4.96|5.09)
Enthusiasm	4.93	6.55	6.59	5.50	6.60	6.57	6.20 (5.39|5.54)	6.29 (5.58|5.70)
Question Handling	4.33	5.77	6.35	6.00	5.60	6.67	6.02 (5.09|5.27)	6.05 (5.21|5.41)
Response Rate							53% (53%|48%)	61% (37%|39%)
Enthusiasm-Pre							4.50 (4.10|4.46)	4.17 (4.36|4.66)
Enthusiasm-Post							4.50 (3.95|4.34)	4.65 (4.25|4.56)
Difference (Enthusiasm-Post − Enthusiasm-Pre)							0.00 (-0.15|-0.12)	0.48 (-0.11|-0.10)

Teaching Assistant Questions:

Presentation	Your teaching assistant presents material in an organized, well planned manner.
Clarity	Your teaching assistant explains concepts clearly and with appropriate use of examples.
Enthusiasm	Your teaching assistant appears enthusiastic and interested in the course material.
Question Handling	Your teaching assistant attends to questions and answers them clearly and effectively.
Overall	All things considered, your teaching assistant performs effectively.

Instructor Questions:

Presentation	The presentation of the material in an organized, well planned manner was
Clarity	The clarity of explanation of the subject matter was
Enthusiasm	The instructor's enthusiasm and interest in the course material was
Question Handling	The attention to students' questions in the classroom and the clarity of the answers was
Overall	What is your overall rating of this instructor as a teacher
Enthusiasm-Pre	Your level of enthusiasm for taking this course, at the time of initial registration
Enthusiasm-Post	Your level of enthusiasm now that you have completed the course

Enthusiasm-Pre/Post Scale: 1. very low, 2. low, 3. below average, 4, average, 5. above average, 6. high, 7. very high

[4] Department of Computer Science, Faculty of Arts and Science
[5] Department of Mechanical and Industrial Engineering, Faculty of Engineering

5.2 Student Comments and Feedback

The comments provided below were included in my evaluations as a teaching assistant and instructor. For the purpose of improving my teaching, I have found periodic in-class feedback (e.g., filling out a 'ticket-out-the-door') from students to be more helpful than the end of term comments. This being said, I did take the feedback I received seriously.

TA Feedback. Tbl. 3 lists the comments I received as a TA, arranged by course. As a result of the feedback I received for *CSC258: Computer Organization*, I focused on my board skills, pacing my explanations, checking in with my learners, and being more informed about the assignment and lecture details.

Table 3: Comments received on TA evaluations (2008–2015).

CSC258: Computer Organization	Fall 2008
"Alicia is the best TA I have had; she is well-prepared, teaches effectively, and appears genuinely excited about the course material." "Very helpful with the examples used in the tutorial." "talks too fast => erases board too fast"	
CSC258: Computer Organization	Winter 2009
"She is fun but sometimes is unable to answer some questions, but generally answers them by the next tutorial. She also goes through the examples and explains them slowly and clearly. The tutorials helped to make clear the material greatly." "Alicia could be a little more prepared for tutorials."	
CSC 209: Software Tools and Systems Programming	Fall 2011
"Very eccentric and enthusiastic, always entertaining."	
CSC 300: Computers and Society	Fall 2015
"I felt the TA was to harsh when marking" "Alicia is great, one of the best TAs I've ever had."	

Instructor Feedback. As an instructor for *APS106: Fundamentals of Computer Programming*, I received positive, detailed, and helpful comments (see Tbl. 4). The students appreciated my enthusiasm, notes, examples, and the respectful space I created. Students wanted more examples, and for me to compile and execute these examples in class. As a first time instructor, I struggled with pacing and understanding what could be expected as prior knowledge. Of particular concern to me, were a few comments I received about showing favouritism. I was seeking to achieve a delicate balance between calling on students directly with the potential risk of embarrassing them, and calling on the subset of students who raised their hands. In 2013, I worked on establishing policies and adding additional activities to my class, so that everyone could participate. I calibrated my expectations and taught to learners who had no programming experience. My pacing was better and only a few students commented that I went too fast. Students still wanted more examples and additional contact time, but overall the feedback I received in 2013 was markedly better. If I taught this course again, I would add a summative example (as one student suggested). I would also hold additional office hours, because they were well attended and both the students and I enjoyed them.

Table 4: Representative Student Feedback from *APS106: Fundamentals of Computer Programming*

APS106: Fundamentals of Computer Programming	Winter 2012
"• very funny • friendly environment • love coming to lectures • goes at a great pace, quick, but more efficient than anything • love her • helpful • always tries very hard to accommodate for the students schedule and learning needs"	
"...The practice problems should [be] interesting programs other than simple programs. Picking on students who are unsure of the material is a little bit embarrassing. There is a sense of favoritism in the classroom. ... Don't assume prior knowledge of computer science from the students."	
"• goes through material too quickly! • but, I like how she is self-aware and encourages feedback + improvement on her teaching skills • AMAZING encouragement of the class involvement + class participation that facilitates learning very well"	
"• very enthusiastic, keeps the class very interesting by adding humor • responds to any questions that the class has • improvement by showing us the code run more often"	
"Thanks for doing examples different from the textbook. Your examples are really helpful."	
"You were by far my favourite prof this year. You always gave students the respect and attention that they deserved and made the lecture hall into a space where people had no fear of asking questions. Your passion and knowledge was very apparent to all of the students. At the beginning of the course I was expecting the worst but you made the whole learning experience so positive for my classmates and I. Thank you sooo much and please keep teaching!!!"	
APS106: Fundamentals of Computer Programming	Winter 2013
"Your enthusiasm for this course really was very infectious. Even when people didn't seem interested you brought interest to the room. Thank you for the dedication to this course."	
"Although I hate programming and have never done it before, you actually made the course interesting and fun to learn. You were very enthusiastic when teaching which made me more into this course. You addressed all the students needs and were very helpful. Good job overall, you were one of the few profs who actually knows how to teach!"	
"I want to thank you for accommodating our needs as students. You did a great job explaining every aspect of this course. The enthusiasm you showed helped keep us interested. Thanks again."	
"You made me excited to program and feel like I could do it when I started the course with no confidence/experience. Now I'm thinking about taking a course in the summer and am excited about programming. Thank you! You taught subjects with clarity and vigour. It was refreshing to have such an excellent and inspiring professor. :)"	
"The format of the lecture (weekly slides which are explain in class) is excellent. More examples are always welcome. A suggestion would be to create a code from the beginning (or the semester) and to add new concepts as it was taught. At the end of every new topic, (end of week) the 'summative' code would contain all the topics so far. An improvement would be to have a greater interaction with the TAs and lab TAs. There is a significant communication gap."	
"Alicia Grubb is an excellent instructor and is definitely very well suited to be a Full Professor. She's always so enthusiastic and very approachable. What makes her unique is that she makes the most dull and boring complicated concepts easy and fun to understand. She keeps the class as involved as possible and even when I'm very behind and tired I still look forward to her class. This course I find can be difficult as I'm a slow learner so as a word of evidence I hope she does more basic examples to help refresh everybody on track. Definitely cares a lot about her students though which I greatly appreciated as I sometimes struggled with course concepts, which is better them my other professors."	
"Alicia Grubb is very likeable, and also an amazing teacher because she makes computer languages easy to comprehend. Also she's a pretty groovy cat. Very useful notes, helpful analogies, and in-depth explanations of examples. The best professor I've had (out of 16)."	
"The instructor was very good overall and provided many opportunities for student feedback and questions. The attention to student questions and ensuring students understand the material was superb."	
"The course material is interesting. The way Alicia teaches is good as you can understand everything clearly. She's available for help too. I would suggest that she increases the office hours during the week"	

6 Diversity Statement

I believe the key to mentoring a diverse student body is to educate oneself about the issues faced by minority groups and broader societal context, and to listen to each student and their diverse perspective. Here I will discuss how my personal relationship with diversity has evolved and the ways I have worked to create change.

Within the context of computer science (CS), I am a member of an underrepresented minority. In my MSc and the early years of my PhD, I was actively involved in the women's group in my department (DCS Women) and mentorship. I also had the privilege of representing the Department at both the CRA-W Grad Cohort (2009, 2010) and the Grace Hopper Celebration of Women in Computing (2009). After meeting such a diverse group of women in CS, I realized that I had been trying to be and act like a man to feel a sense of belonging in the CS community. It was during this period that I took and audited several courses in sociology and political science (i.e., globalization, human development, international trade, and social network theory), while engaging with queer and gender theory discussion groups. I began to engage with the concept of equity and the historical and systematic oppression of people. Taking off the blinders, I recognized the persistent prejudice and biases that still exist today. I engaged in broader conversations about colonialism, the patriarchy, ableism, racism, classism, homophobia, and transphobia, and how people are affected by the intersectional nature of their identities. I learned that to address structural inequalities, there needs to be both structural (i.e., institutional) change and a shift in the unconscious biases and resulting behaviour of everyone. I finally acknowledged my own unconscious bias against women and other minorities in STEM, and in 2014, I made a commitment to explore my authentic self and value the diverse experiences of those around me. Since then, I fundamentally think differently about my own actions, my teaching, and my research. I believe a just society must **give each member the support they need to achieve their maximum potential** (as described in John Rawls' A Theory of Justice).

In my teaching, I spent significant time thinking about how lectures and assignment examples may exclude individuals by presuming that CS students had common interests (i.e., the geek culture, video games, and other cultural references). I also thought about how many students are unduly influenced in examinations by priming and stereotype threat. In my lectures, I became cognizant of the power of language and the importance of addressing hate speech. I also became aware of who was participating and started using active learning techniques to encourage as many learners to participate as possible. In curriculum development, I thought about how access to a computer and early programming experience advantaged some students, while cultural and societal expectations disadvantages others (e.g., first in the family). In choosing undergraduate research assistants, I had to check my own biases to ensure that I wasn't choosing students who had similar interests to me, and ensure that I evaluated their experience relative to when they discovered CS.

I have also focused on making subtle **changes that can impact all members of my department**. I volunteered to coordinate and teach the training for incoming CS teaching assistants (TAs) in 2014 and 2015 to ensure that topics of harassment, micro-aggression, and hurtful language were introduced and discussed. The goal was to empower TAs to voice opposition to such inappropriate behaviour. 180 TAs attended my training sessions. In their feedback TAs consistently reported valuing the 'scenario part' where we included these topics. For example, one TA wrote that the scenarios "brought to [their] attention issues that [they] had not thought about before". In the Fall 2015 term, I advocated for the introduction of an Equity in Computer Science module in *CSC300: Computers and Society*. I designed the module and presented it in the first year (my slides are available online). The lecture explores structural inequalities within CS education. Thus far around 500 students have now been exposed to the concept of equity through CSC300.

The activities in the previous paragraph were not directly visible to other students. To **increase such visibility**, I served on two 'Women in CS' Panels: one at the 2017 Canadian Undergraduate Computer Science Conference and one for the undergrad Women in CS Club (2017). As a result, I have received multiple requests for mentorship and unexpected gratitude for my honesty and insights. I chose to work as a TA in the First Year Help Centre over the last year, specifically to be visible as a woman in CS and to lend my support. At a broader advocacy level, I was a member of the Engineering Positive Space Committee and organized Pink T-Shirt Day (an anti-bullying initiative) in 2014 & 2015. I also volunteered to give a work history testimonial as part of the Engineering Instructional Innovation Project (2014–2015) so that first year students could learn from the negative experiences I faced in industry. Finally, I have informally mentored several peers and encouraged them to explore their own assumptions and bias. One later wrote, "Alicia Grubb taught me many things, most crucially to examine critically my own conduct and behaviour" [Famelis, 2016, p. iv]. I fully support women-only spaces as a way to nurture women already in STEM and have continued participating and contributing to DCS Women. I am proud to have worked with and learned from a number of outstanding individuals. While there have been some setbacks and there will be future challenges, I intend to **continue working toward equity in CS** and a just society.

7 Sample Teaching Materials

7.1 Lecture

The examples in this section come from my work as an instructor for *APS106: Fundamentals of Computer Programming.* The course objective was to teach students how to program in C, but I also saw this course as a way to teach students about problem solving and algorithms. APS 106 was a required course for first year chemical engineers and many students felt it was outside the scope of their program. In an attempt to increase engagement, I used examples from a variety of domains including chemical engineering. To set the tone for the term, I used both these examples in the first two weeks of class when I was teaching data types, arithmetics (i.e., arithmetic, rational, and logical operators), formatting input and output (i.e., scanf and printf).

Kg-Moles-Atoms Example.
Students really liked the Kg-Moles-Atoms example. I liked it because I could use it later in the term when I introduced reading from a file, where the goal was to read the element number, name, and molar mass from a file.

Write a program in C to help users convert from mass in Kg to moles and number of atoms.

Your program should have the following sample output for Gold:

```
Input the mass in Kg: 2.0
Input the molar mass (g/moles): 196.97
2.00 Kg of your substance is the same as 10.1538 mole.
2.00 Kg of your substance has 6.1146e+024 atoms in it.
```

Equilibrium Constant (K_T) Example.
When I introduced the Equilibrium Constant example, which uses the Habor-Bosch process (a fundamental example in physical chemistry), some of my learners were hostile to my expectation that they would remember content discussed in a previous term. I repeatedly assured them that the material was to showcase how they could use programming concepts, and that they didn't actually have to remember the process because all required information was provided. Despite their initial resistance, I believe these examples helped me engage my learners.

Write a program in C that calculates the Equilibrium Constant (K_T) for the Haber-Bosch Process ($N_2 + 3H_2 \Leftrightarrow 2NH_3$) at a given temperature (in C).

Your program should have the following sample output:

```
Input the temperature in C: 20.0
The Equilibrium Constant for the Haber-Bosch Process at 20.0 degrees Celsius is 7.2797e+05.
```

Notes:
Gibbs Energy $G = -RT \ln K$ (solve for K).
TEMP_C_TO_K = 273.15
GIBBS_ENERGY = -32.9E3
GAS_CONSTANT = 8.3145

7.2 Evaluation/Examination

The examples in this section come from my work as an instructor in 2012 and 2013 for *APS106: Fundamentals of Computer Programming*. The course objective was to teach students how to program in C. APS106 was a required course for most first-year engineers. In 2012 I created part of the midterm exam and part of the final exam. In 2013, I took on a leadership role in creating the midterm exam and did not work on the final exam.

2012 Final Exam Q4.
The primary objective of Q4 is to test the students ability to merge two arrays creating a third. It also tested their ability to use loops and format printed text. This question combined understanding code and writing code. Since I knew that I would be marking over 500 answers, I focused on making it straight forward and clear. Generally students understood the question. In my original marking scheme, I had to consider both *(z+1) array notation as well as z[i] array notation. Solution in red.

This program starts with two sorted int arrays, x and y. The program then merges the two sorted arrays into a third sorted array z. Assume there are no duplicates in arrays x and y.

The program output is:
The first array is: 5 22 26 41 57 60 77 79 95 100
The second array is: 0 15 23 25 42 58 67 78 86 99
The merged array is: 0 5 15 22 23 25 26 41 42 57 58 60 67 77 78 79 86 95 99 100

Consider the following code:

```
#include <stdio.h>

void printArray(int size, int parray[size]);
void merge(int xsize, int ysize, int x[xsize], int y[ysize],
                int z[xsize + ysize]);

int main(void)
{
        int xsize = 10;
        int ysize = 10;
        int zsize = xsize + ysize;

        int x[] = {5,22,26,41,57,60,77,79,95,100};
        int y[] = {0,15,23,25,42,58,67,78,86,99};
        int z[zsize];

        printf ("The first array is:  ");
        printArray(xsize, x);
        printf ("The second array is: ");
        printArray(ysize, y);

        merge(xsize,ysize,x,y,z);

        printf ("The merged array is: ");
        printArray(zsize, z);

}
```

Question continues on the next page.

2012 Final Exam Q4 continued.
Solution in red.

PART A [3 marks]: Write the function printArray in the space provided.

```
void printArray(int size, int parray[size])
{
        int k;
        for (k = 0; k < size; k++)
                printf("%d ", parray[k]);
        printf("\n");
}
```

PART B [5 marks]: Write the function merge in the space provided.

```
void merge(int xsize, int ysize, int x[xsize], int y[ysize],
        int z[xsize + ysize])
{
        int i;
        int x1 = 0;
        int y1 = 0;
        for (i = 0; i < xsize+ysize; i ++){
                if (x1 == xsize)
                        *(z+i) = *(y+(y1++));
                else if (y1 == ysize)
                        *(z+i) = *(x+(x1++));
                else if (*(x+x1) < *(y+y1))
                        *(z+i) = *(x+(x1++));
                else
                        *(z+i) = *(y+(y1++));
        }
}
```

2013 Midterm.
I was unsatisfied with the 2012 Midterm exam because I felt it did not cover sufficient breath of the course content. In 2013 I advocated for a wide range of question, including the three presented here (Q1, Q5, and Q7). There were eight questions on the midterm.

2013 Midterm Exam Q1.
Q1 tests their ability to convert numbers between binary and decimal. I felt this was an important concept to include on the midterm, because later in the course we would talk about memory allocation and address space.

Part A [1 mark]: Convert the following binary number to decimal (base 10): 0100100110

284

Part B [1 mark]: Convert the following decimal number to hexadecimal (base 16): 908

38C

2013 Midterm Exam Q5.
Q5 specific test and reinforces the students ability to connect while loops with for loops. Students found it easier to convert from a for loop to a while loop so we tested them in the opposite direction. This was considered a short answer question.

[2 marks]
Rewrite the following while loop as a for loop:

```
int i = 1;
int j = 1;
while (i <= 15){
        j *= i;
        printf("i is %d and j is %d\n", i, j);
        i++;
}

// Enter solution below.
int i;
int j = 1;
for (  i = 1; i <= 15; i++  )   {
        j *= i;
        printf("i is %d and j is %d\n", i, j);

}
```

2013 Midterm Exam Q7.
Q7 was one of two challenging questions on this midterm. It tested the students ability to write a complete C program and to construct a small algorithm. Specifically the question tested their ability to use printf and scanf statements, as well as loops and arithmetic operators.

[5 marks]
Write a complete C program that prompts the user for a positive integer using the message **"Give a positive integer:"** and computes the sum of all the odd integers between zero and the provided integer (inclusive).
An odd number is an integer which is not a multiple of two (e.g., 1, 3, 5, ...).
Your program should output two lines (other than the prompt):
- the first line prints the odd integers, each separated by a space
- the second line prints the sum of the odd numbers

Here is an example output from the execution of the program:

```
Give a positive integer: 10
1 3 5 7 9
25
```

```
#include <stdio.h>
int main ( ) {

        int num, n = 0, sum = 0;

        // Prompt user and read input
        printf("Give a positive integer: ");
        scanf("%d", &num);

        // Check all numbers up to num
        while(n <= num){
                // Check if it is odd
                if( n % 2 != 0){
                        printf("%d ", n); // Print it
                        sum = sum + n; // Sum it up
                }
                n++;
        }
        printf("\n");
        printf("%d\n", sum); // Print the sum
        }
        return
```

7.3 Syllabus

In this section I present three syllabi. The first two syllabi are from my supervision of research students taking *CSC494:Computer Science Project*. This course is meant to give students a closer look at computer science research and involves any significant project in a area of CS. I co-create each syllabus with the individual student, adapting it to their goals and needs. The syllabus then works as a rubric for evaluating the student at the end of the term.

The first syllabus *CSC494 Fall 2017: BloomingLeaf Tool Extension Project Course Outline* was a coding-based course. This student had concerns about their presentation skills and writing abilities. I agreed to allow them to contribute wiki pages to the project rather than to the original report. We both felt this would help the student focus on the type of writing they might do in industry.

I included the second syllabus *CSC494 Fall 2017: Evolving Goal Models Course Outline*, because it is an example of how I would structure a research-based syllabus for a student who wants to do empirical methods. This student wanted to do a case study comparing two research methods. This syllabus really helped keep the project in check because I had included all the tasks needed to complete the study and a general order of completion.

The third syllabus (4 pages) is a more traditional syllabus and comes out of my work as a teaching assistant (TA) for *CSC300: Computers and Society*. After grading the written assignments and term papers for which the quality varied dramatically, I recognized the potential to engage students in writing and critical thinking. The following term a new instructor (Mathew Zaleski) took over the course and I asked if I could spearhead a curriculum development effort. The syllabus on the following page is the joint effort of Dr. Zaleski and I.

I included this syllabus because it showed how we carefully structured the course. I scaffolded the assignments, such that each assignment was built on the previous one. Page 2 shows this scaffolding and our transparency for when assignment handouts would be available. We introduced writing, public speaking and debate into the tutorial exercises to give learners structured practice. Page 4 of the syllabus, shows the timeline for the term and what would be covered in each tutorial. This syllabus worked as an excellent skeleton for later offerings, where we offered students the opportunity to resubmit their work and also told them when each of their assignments would be returned.

CSC494 Fall 2017: BloomingLeaf Tool Extension Project Course Outline

Course Description

This course involves improving the functionality and implementing new features/extensions for a pre-existing software tool with a large code-base, "BloomingLeaf". The BloomingLeaf tool allows users to draw and analyze goal-oriented requirements models with temporal elements as part of a model-based Software Engineering Methodology. Student(s) will work on one of several projects related to BloomingLeaf: expanding the ability of the tool to facilitate and visualize reasoning, adding assignment of multiple time points in a path, allowing multiple views over a single model, and improving the overall functionality/usability of the tool. By the end of the course students will have gained experience in the following:

- Working with a large code base
- Working with a git and the ticketing system in github
- Interpreting and expanding on requirements
- Evaluating User Interfaces
- Several technologies:
    - JavaScript and clientIO Libraries
    - CGI with Python
    - HTML, CSS, and other web-based technologies

Course Evaluation

15% Participation:
Participating will include attending weekly meetings, devoting at least ten hours per week to the project, using the bug system, and being reasonably available via electronic communication.

45% Progress:
40% Contributions: Students will be evaluated on their contributions to the code-base, keeping in mind the learning curve that comes with working with a large body of pre-existing code.
5% Weekly Updates: Students will be asked to provide written weekly updates and document their goals and progress.

5% Testing:
Students will be expected to provide the means to test their progress, either through automated unit tests, developing a set of test examples, or written GUI test instructions.

15% Documentation:
Students will also be asked to add sufficient documentation to code via comments. Students will be required to create instructional videos for the tool and their features.

20% Reports:
Students will be asked to prepare two reports:
5% Requirements Description: A two-page analysis of the tools' usability with suggestions on how to improve the tool and the details of which updates will be included in this project. Due: Last day of September.
15% Final Report: Add five to six-pages to the GitHub wiki, describing in detail how to use the tool and each feature, as well as the projects progress, plan changes, and obstacles. This can include a modified version of the Requirements Description. Due: Last day of classes in December.

CSC494 Fall 2017: Evolving Goal Models Course Outline

Course Description

This course involves studying techniques for understanding how the evaluations of goals in a goal model change over time. Two tools and their respective techniques will be investigated. GrowingLeaf uses conventional iStar (or i*) notation and allows for multiple types of analysis. BloomingLeaf uses Tropos like semantic with the visual notation of iStar intentions to perform constraint-based analysis.

The course project involves doing a comparison between these two tool as well as conducting a full user-study to the impact of the change in the underlying semantics, asking questions such as "Does the transformation from iStar to Tropos make things more reasonable?"

By the end of the course students will have gained experience in the following:

- Working with two large code bases
- Designing, conducting, and analyzing an empirical study
- Completing the protocol submission for the Research Ethics Board at the UToronto
- Working with a gitHub and Slack
- Interpreting and expanding on requirements
- Evaluating User Interfaces
- Iterative writing of research findings
- (Optional) Student may gain exposure to several technologies:
    - JavaScript and clientIO Libraries
    - CGI with Python
    - HTML, CSS, and other web-based technologies

Course Evaluation

10% Initial Evaluation: A one to two page description of the requirements for the specific project they will undertake. The student should show understanding of the tools and techniques and list/analyze their differences.

10% Study Design Document: Specify research questions, study protocol, and a discussion of what data will be collected in the study and how it will be analyzed. Including a brief literature review.

10% REB Protocol Submission: Complete the protocol submission process and approval.

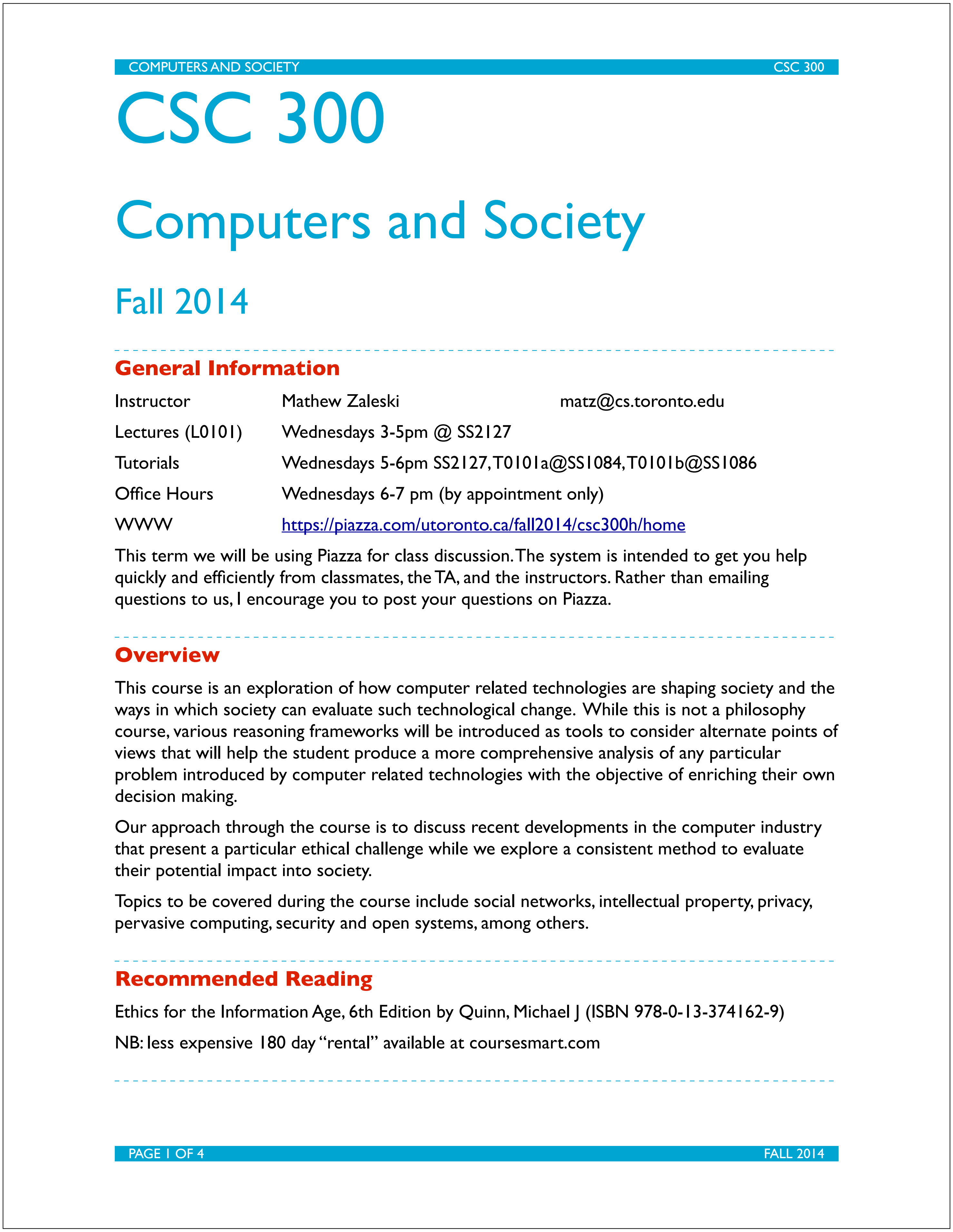
10% Study Materials: All questionnaires, videos, letters, posters, and any other study materials.

20% Participant Observation and Recruitment: Students will be graded on how they conduct the in-person portion of the student and their ability to recruit participants.

20% Study Analysis and Write-Up: Students will provide an complete analysis of study data, and results should be written up for a workshop submission.

15% Participation and Professionalism: The participating will include attending weekly meetings, and being reasonably available via electronic communication.

5 % Progress Updates: Students will be asked to provide written weekly updates and document their goals and progress. Students will also be asked to reflect on their experiences in the course.



CSC 300

Computers and Society

Fall 2014

General Information

Instructor	Mathew Zaleski	matz@cs.toronto.edu
Lectures (L0101)	Wednesdays 3-5pm @ SS2127	
Tutorials	Wednesdays 5-6pm SS2127, T0101a@SS1084, T0101b@SS1086	
Office Hours	Wednesdays 6-7 pm (by appointment only)	
WWW	https://piazza.com/utoronto.ca/fall2014/csc300h/home	

This term we will be using Piazza for class discussion. The system is intended to get you help quickly and efficiently from classmates, the TA, and the instructors. Rather than emailing questions to us, I encourage you to post your questions on Piazza.

Overview

This course is an exploration of how computer related technologies are shaping society and the ways in which society can evaluate such technological change. While this is not a philosophy course, various reasoning frameworks will be introduced as tools to consider alternate points of views that will help the student produce a more comprehensive analysis of any particular problem introduced by computer related technologies with the objective of enriching their own decision making.

Our approach through the course is to discuss recent developments in the computer industry that present a particular ethical challenge while we explore a consistent method to evaluate their potential impact into society.

Topics to be covered during the course include social networks, intellectual property, privacy, pervasive computing, security and open systems, among others.

Recommended Reading

Ethics for the Information Age, 6th Edition by Quinn, Michael J (ISBN 978-0-13-374162-9)

NB: less expensive 180 day "rental" available at coursesmart.com

Prerequisites

There are no particular prerequisites for this course, but every student will be expected to participate verbally throughout the term as well as write short essays with a solid argument around a number of topics, so we expect a good level of proficiency in the English language.

Students are encouraged to take advantage of their college writing centres.
See http://www.writing.utoronto.ca/writing-centres

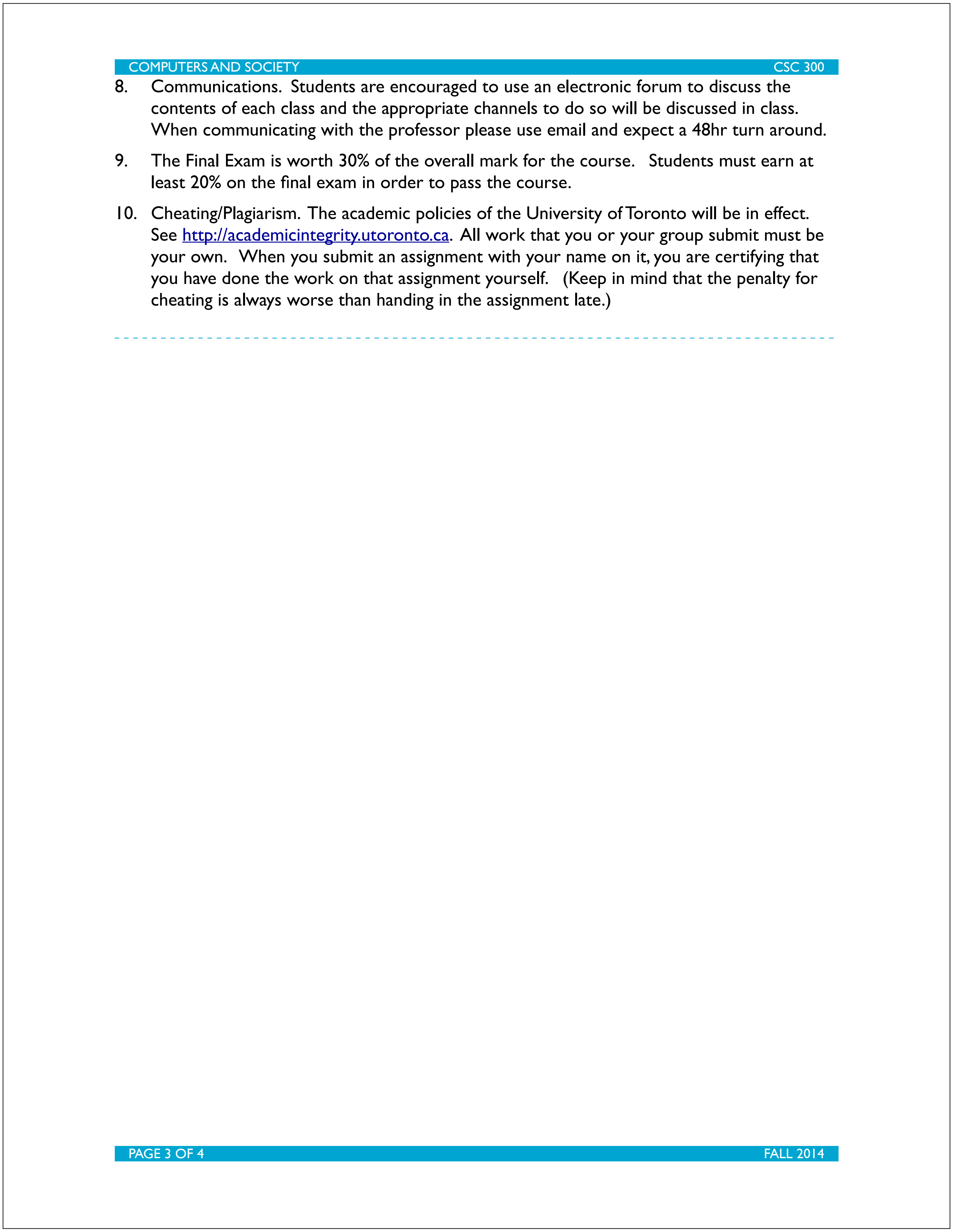
Marking

Course grade will be determined as follows:

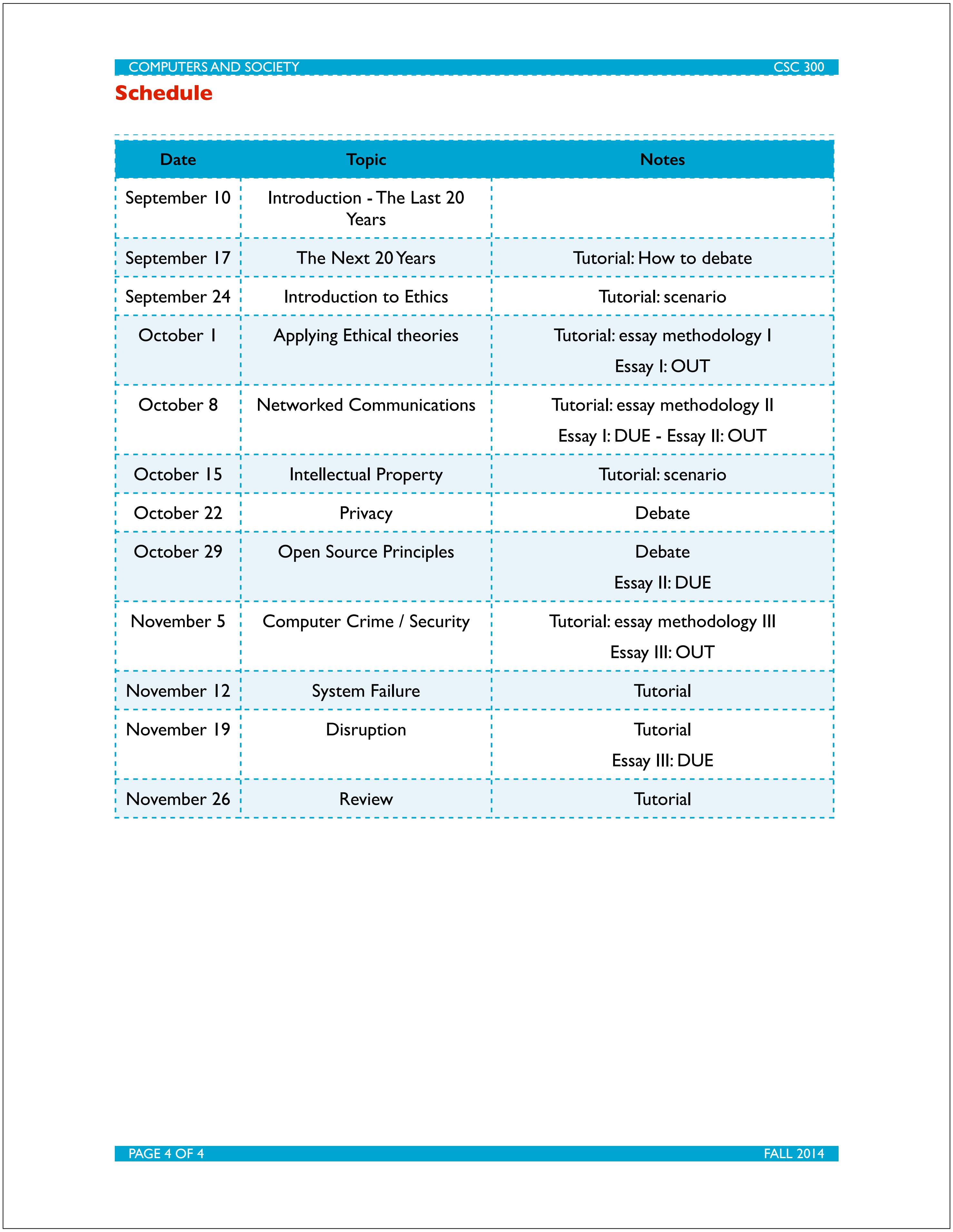
	Weight	Out	Due
Participation	0.1		
Mid-Term Debate	0.2		Oct 22/29
Essay I - Proposal	0.1	Oct 1	Oct 8
Essay II - Ethical Framework	0.15	Oct 8	Oct 29
Essay III - Final Report	0.15	Nov 5	Nov 19
Final Exam	0.3		

Assignment/Project Policies

1. Participation mark is earned by preparing each class using the reading list.
2. The Mid-Term Debates will be done in groups of 3 to 5 students. Each group will be assigned a topic and a position to defend during the debate. All members of the group must participate in the debate to earn a mark.
3. All the essays are to be prepared individually. The essays will be based on a term-long project that will explore an issue chosen by each student and analyzed according to the methodology learned during class.
4. Essays are due at midnight on their due date.
5. All deliverables should be submitted electronically.
6. **Late course work**. No assignment will be accepted after the deadline.
7. **Re-marking**. The TA's will hold a special re-marking session shortly after assignments are handed back to students. If you are still dissatisfied after talking to the TA, then set an appointment with the instructor. All re-marking should be complete within one week of the date when the marked assignment is available for hand-back. No assignment will be re-marked after this period.



8. Communications. Students are encouraged to use an electronic forum to discuss the contents of each class and the appropriate channels to do so will be discussed in class. When communicating with the professor please use email and expect a 48hr turn around.
9. The Final Exam is worth 30% of the overall mark for the course. Students must earn at least 20% on the final exam in order to pass the course.
10. Cheating/Plagiarism. The academic policies of the University of Toronto will be in effect. See http://academicintegrity.utoronto.ca. All work that you or your group submit must be your own. When you submit an assignment with your name on it, you are certifying that you have done the work on that assignment yourself. (Keep in mind that the penalty for cheating is always worse than handing in the assignment late.)



Schedule

Date	Topic	Notes
September 10	Introduction - The Last 20 Years	
September 17	The Next 20 Years	Tutorial: How to debate
September 24	Introduction to Ethics	Tutorial: scenario
October 1	Applying Ethical theories	Tutorial: essay methodology I Essay I: OUT
October 8	Networked Communications	Tutorial: essay methodology II Essay I: DUE - Essay II: OUT
October 15	Intellectual Property	Tutorial: scenario
October 22	Privacy	Debate
October 29	Open Source Principles	Debate Essay II: DUE
November 5	Computer Crime / Security	Tutorial: essay methodology III Essay III: OUT
November 12	System Failure	Tutorial
November 19	Disruption	Tutorial Essay III: DUE
November 26	Review	Tutorial

8 Sample Training Materials

8.1 First-Year Teaching Assistant Training

I worked as a Teaching Assistant (TA) trainer for the Department of Computer Science (2014–2015). I coordinated and led the TA Training for all incoming CS TAs in a 3-hour workshop broken down into two time slots. This workshop covered active learning techniques, running tutorials/labs, getting/giving feedback, marking and plagiarism, and professionalism and harassment. Each training session had approximately 40 attendees.

Marking Activity. When evaluating student answers there were often multiple correct answers (although usually one was considered the best answer). Some students made individual mistakes in an answer but other students came up with unique ways of answering. In working with both undergraduate and graduate teaching assistants, I observed that many TAs had trouble evaluating answers that were incorrect in a non-standard way. Since most TAs excelled in the courses they were working with, they were unaware of the variety of student responses and were dismissive of these non-standard responses. I hoped to demonstrate that these non-standard answers could convey partial understanding deserving of part marks, even if the initial rubric did not clearly allow for it. By the end of this activity, I hoped that TAs will be able to: (a) apply the given marking scheme, (b) differentiate between standard and non-standard answers, and (c) identify how non-standard answers vary from the given marking scheme.

The marking activity (see Department of Computer Science TA Training - Session 2) was placed in the last hour of training and was planned to take 30 minutes. After giving a presentation on both the structural and technical details of how to grade assignments in the department, my co-trainer and I gave the TAs the opportunity to practise grading. TAs organized themselves into groups of 3 or 4, and created a marking scheme. Each TA read and individually graded all answers on the handout. Then the TAs were given time to discuss and compare grades. Lastly, as a whole group again, we picked a few answers and asked everyone (by a show of hands) what grade they gave for a given answer, and took questions. Note: We told the TAs that they would get 10 minutes to grade individual answers, but we actually gave them 15 minutes. At the end of the activity we pointed out to them that they took 15 minutes and grading took much longer than they expected.

Skill Development

- practice following a rubric and reviewing student answers
- learn how to approach and evaluate non-standard answers
- learn from peers how to improve evaluation

Benefits

- allows TAs an opportunity to practice grading
- give TAs an awareness of budgeting time to grade answers
- allow TAs to compare answers and understand how subjective grading can be
- foster a collaborative approach to grading within courses

Challenges

- issues arising from TAs not knowing whether they will be grading code or proofs
- issues arising from TAs being reluctant to form groups
- concerns/anxieties of TAs about sharing their evaluations

Activity Improvements. The handout included is from the first-time we used this activity. In subsequent offerings we only used *Activity 2: Grading and giving feedback* and removed *Activity 1: Creating a marking scheme.* I felt that having the TAs create a rubric distracted from the point of the activity. Also, first-time TAs were rarely asked to make rubrics and so it was deemed unnecessary for training.

Note: The Session 2 handout was developed with my co-trainer Brian Law.

Department of Computer Science TA Training - Session 2

ACTIVITY 1: CREATING A MARKING SCHEME

Create a marking scheme (in the space below) given the following advice and Question:

Advice: “Mark this question out of five, mark only for correctness and programming style, and make sure they handle all the special cases.”

Implement the following function, according to its docstring: (5 marks)

```
def truncate(string1, substring1, i, substring2):
""" (str, str, int, str) -> str
This function finds the i-th occurrence of substring1 in string1, and then truncates string1 after
that occurrence, replacing everything after the i-th occurrence with substring2.

If there is no i-th occurrence of substring1 in string1, the function should return the original
string1 unchanged.

Example:
>>> truncate("Today I will study and eat and sleep.", "nd", 2, " study some more.")
"Today I will study and eat and study some more."
"""

index = -1
for counter in range(i):
       index = string1.find(substring1, index+1)
       if index == -1:
           return string1
if index != -1:
       index += len(substring1)
       string1 = string1[:index] + substring2
return string1
```

MARKING SCHEME:

ACTIVITY 2: GRADING AND GIVING EFFECTIVE FEEDBACK

In the space provide grade each of the answers below using your making scheme from Activity #1. Write down some useful feedback for the student.

```
counter = 0
ith_index = 0

while counter < i:
    ~~ith_index += string1[ith_index].find(substring1)~~
    ith_index += string1[ith_index + len(substring1) :].find(substring1)
    counter += 1
ith_index = ith_index + counter    #Accounts for starting
                                   # counting at 0 each time
                                   # we go through the loop

~~if i >= 0:~~
if ith_index >= 0
    ~~output_str = string1[: ith_index] + substring2~~
    output_str = string1[: ith_index + len(substring1)] + substring2
else:
    output_str = string1

return output_str
```

```
str_list = string1.split( )
item_list = [ ]
for item in str_list:
    if item == substring1:
        ~~if str_list.index(item) =~~
        item_list.append(item)
        for ~~item~~ ~~i in range (len(item_list):~~
            ~~if~~

~~return~~
for item in item_list:
    if item_list.index(item) = i-1
        ~~re~~string = string1 ~~E~~.replace (item[i]: ) - substring 2
        ~~string1. replace (item i: ) = substring2~~

return string ~~1~~
```

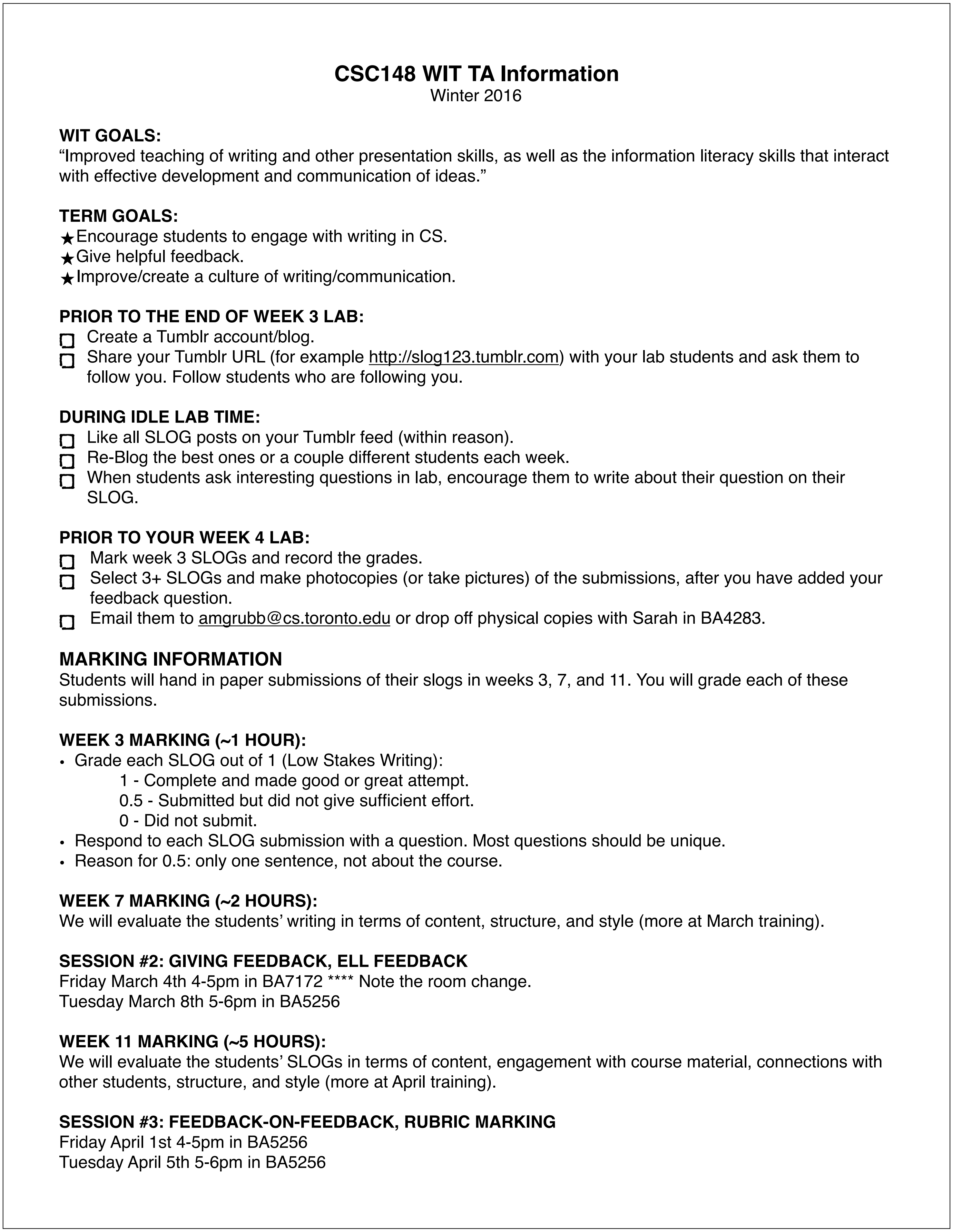
```
if string1.count(substring1) < i
    return string1
list1 = string1.split(substring1)
while i < len(list1)
    list1.pop()
list1.append(substring2)
new_string = substring1.join(list1)
return new_string
```

```
string1_final = " "
~~string1 = string1.split("substring1")~~

~~[illegible]~~
~~[illegible]~~
list1 = string1.split("substring1)
del list1[i]
list1.insert(i,substring2)
~~[illegible]~~
for item in list1:
    if item != list[-1]:
        string1_final += item + ~~[illegible]~~ substring1
    else:
        string1_final += item
return string1_final
```

8.2 Writing Teaching Assistant Training

I worked as the Computer Science Writing Fellow (2014–2016) as part of a broader Writing Across the Curriculum program, locally called WIT (Writing Instruction for Teaching Assistants). The sample material in this section are both from my course specific training session for *CSC148: Introduction to Computer Science* in January 2016. In this session I introduced the goals of WIT, and talked about Using Rubrics and Low Stakes Writing. I used a slide deck (not included but available upon request) to introduce concepts and display the course rubric. The two documents that follow are the handout I gave to participants in the training session, entitled *CSC148 WIT TA Information*, and my summary report of the training session, entitled *CS WIT Training Session: Participant Feedback*. I included the handout as an example of how I clearly communicate my expectations to my TAs. The handout also lists the plans for subsequent training sessions and dates. After each training session, I created a report of my overall impressions of the workshop and a summary of the participants' feedback. By writing these reports after each session, I could look back on them before preparing the next workshop. They also helped when I transitioned my training responsibilities to others. I included this report as an example of the reports I write. They have proved valuable and I plan to continue writing them for future training sessions.



CSC148 WIT TA Information

Winter 2016

WIT GOALS:
"Improved teaching of writing and other presentation skills, as well as the information literacy skills that interact with effective development and communication of ideas."

TERM GOALS:
★Encourage students to engage with writing in CS.
★Give helpful feedback.
★Improve/create a culture of writing/communication.

PRIOR TO THE END OF WEEK 3 LAB:
☐ Create a Tumblr account/blog.
☐ Share your Tumblr URL (for example http://slog123.tumblr.com) with your lab students and ask them to follow you. Follow students who are following you.

DURING IDLE LAB TIME:
☐ Like all SLOG posts on your Tumblr feed (within reason).
☐ Re-Blog the best ones or a couple different students each week.
☐ When students ask interesting questions in lab, encourage them to write about their question on their SLOG.

PRIOR TO YOUR WEEK 4 LAB:
☐ Mark week 3 SLOGs and record the grades.
☐ Select 3+ SLOGs and make photocopies (or take pictures) of the submissions, after you have added your feedback question.
☐ Email them to amgrubb@cs.toronto.edu or drop off physical copies with Sarah in BA4283.

MARKING INFORMATION

Students will hand in paper submissions of their slogs in weeks 3, 7, and 11. You will grade each of these submissions.

WEEK 3 MARKING (~1 HOUR):
- Grade each SLOG out of 1 (Low Stakes Writing):
  - 1 - Complete and made good or great attempt.
  - 0.5 - Submitted but did not give sufficient effort.
  - 0 - Did not submit.
- Respond to each SLOG submission with a question. Most questions should be unique.
- Reason for 0.5: only one sentence, not about the course.

WEEK 7 MARKING (~2 HOURS):
We will evaluate the students' writing in terms of content, structure, and style (more at March training).

SESSION #2: GIVING FEEDBACK, ELL FEEDBACK
Friday March 4th 4-5pm in BA7172 **** Note the room change.
Tuesday March 8th 5-6pm in BA5256

WEEK 11 MARKING (~5 HOURS):
We will evaluate the students' SLOGs in terms of content, engagement with course material, connections with other students, structure, and style (more at April training).

SESSION #3: FEEDBACK-ON-FEEDBACK, RUBRIC MARKING
Friday April 1st 4-5pm in BA5256
Tuesday April 5th 5-6pm in BA5256

CS WIT Training Session: Participant Feedback

Department:	Computer Science	Trainers:	Alicia Grubb
Session Title:	WIT January Training	Duration:	[1 hour] x 2 groups
Number of participants:	24	Dates:	January 22 & 26, 2015

Quantitative Data.

General rating of the workshop (useful ideas about and/or strategies for teaching writing):

1. poor	2. fair	3. good	4. very good	5. excellent
0	0	4	8	12

Average: 4.33

Which aspect(s) of the workshop did you find most useful?

All of it. **
 The SLOG has changed a lot so this is a good overview.
Knowing the purpose of WIT.
 Explaining how to cultivate a culture of writing.
 The focus on the importance of writing.
 Idea of "low stakes" writing.
 Pedagogical reasons / studies / statistics.
Instructions & Tips on how to approach the SLOGs & how to respond to them & grade them.
 Going through examples of how to read, encourage, and respond to students slogs. **
 Marking scheme and how exactly is SLOG organized. **
 Coming up with questions to ask to student's posts. *****
 Analyzing past SLOGs.
How to Tumblr / Setting up Tumblr ****

Were there any aspects of the workshop you think should be changed or revised? If so, how?

Greatest workshop I've ever been to.
Have more time for activities.
More time with examples and more examples of students SLOGs. ***
Maybe give some reasons for why blogging is fun for students?
Don't need motivations for making Tumblr.
TAs could create Tumblr accounts and their own time.
More careful walk through expectations (liking, re-blogging).

How will you use what you learned today to help engage your students in writing?

- Learned how to encourage students to participate in the SLOGs.
- I will inform them of the applications/usage.
- Get them excited about slogging, interact with them online, get them motivated by providing questions, comments & likes.
- I will try to engage them and make them excited about writing.

- Use a similar approach to devising questions as we did today.
- Use my TA slog extensively to motivate them.
- Showing enthusiasm and encouragement.
- Focus on motivating them to write often for communication.
- I will ask insightful questions & encourage students to build our own community.
- Be as enthusiastic as possible so they are in turn excited to learn.
- Be excited and focus on creating community.
- Knowing most don't think it's fun, will make effort to show them how fun it is.
- I'll try to participate in the tumblr thing.
- Encouraging students to write SLOGS related to questions they have in lab to help build community.

Please suggest areas you'd like additional training in, related to writing instruction.

How to evaluate technical writing specifically. ****
How to evaluate content if the writing is not well written. How much should the penalty be based on poor writing vs. the idea? / How do I mark people who are fluent and non-flint on the same level?
What constitutes a good / useful CS blog post.
How to deliver constructive criticism.
I'd like to get training on methods of encouraging students to write more enthusiastically. I feel that their SLOGs are forced.
Writing for specific audiences.
Commenting on quizzes, explaining where they went wrong, etc.
Leading labs.

Facilitator Notes:

The first session had 6 participants. Two were returners and needed space to air grievances from last year. This resulted in us being a bit rushed for time. One participant had a lot of trouble grasping the concepts and rubric for grading the SLOGs.

The second session went slightly faster than expected, everyone seemed really engaged and on board. I felt very relaxed at this session.

I think my slide deck has just enough intro to motivate WIT.

On a whim, I decided to talk about low-stakes writing as a metaphor for encouraging students to blog. I think this was a really great idea and this was new material for all participants. In the first session, one participant could not make the connection that it was a metaphor, and thought we were doing it all wrong. In the second session, participants seemed excited and intrigued that they had never heard of low-stakes writing before.

My intuition after these two training sessions is that we should never start training by saying "we are going to go fast through the material", even if that is the case. Instead just say "there is enough time for everything", even if we know we are going to go through it pretty fast.