CSCI-400: Principles of Programming Languages

Spring 2019: Course Syllabus

Instructor:	(Jonathan) Sumner Evans (jonathanevans@mines.edu)
TA:	Ben Tarman (btarman@mines.edu)
Course Adviser:	Christopher Painter-Wakefield (cpainter@mines.edu)[1]
Lecture:	Tuesdays and Thursdays, 9:30 AM to 10:45 AM in BB W250
Office Hours:	See the course website for a listing of the current schedule. Alternatively, you may email to schedule an appointment, or just schedule a block of time on my Google Calendar here: https://goo.gl/ZJdSfc.
Course Website:	https://lambda.mines.edu
Piazza:	https://piazza.com/mines/spring2019/csci400/home
Prerequisites:	CSCI-262 (Data Structures) and CSCI-306 (Software Engineering)[2] *Optional:* CSCI-274 (Intro to Linux) – see Expectations for more info.

Course Description & Learning Objectives

CSCI-400 is a course covering **the design and implementation of computer programming languages**. This topic is highly relevant for both future software engineers and computer scientists: by exposing oneself to how programming languages are made, software engineers will be able to learn new programming languages in their profession to adapt to rapidly evolving technology, and computer scientists will explore a practical usage of computational theory.

By the end of this course, students should be able to:

1. Explain common programming language concepts, such as evaluation order, typing systems, memory management, paradigms, scoping, tail recursion, etc., and identify their usage in industry programming languages.
2. Use mathematical foundations of the lambda calculus to define programming language constructs.
3. Understand language-oriented programming by implementing a domain-specific programming language.
4. Implement an interpreter for a non-trivial programming language.

To fulfill these learning goals, we will use numerous programming languages during the course, including many dialects of the Racket programming language, and Python. Students will be given multiple projects and homework assignments that relate directly to these goals.

[1] You probably won't ever need to contact the course adviser, but if you have concerns you wish not to bring to the attention of the instructor (e.g., anonymous concerns), you are welcome to contact the adviser.

[2] If you have prior experience in Java or other OOP languages, I may (at my own discretion) waive this prerequisite. Please contact me if you are interested in pursuing this option.

Expectations

Students are expected to develop all code for this course using a Linux machine. Basic familiarity with the Linux command line is assumed: CSCI-274 (Intro to Linux) is highly recommended (even as concurrent enrollment) if the student is not already familiar.

Students are welcome to develop either on their own Linux computers, or at the Linux lab provided by the school: ALAMODE. ALAMODE is located at BB 136 and you can login using MultiPass credentials.

Students are expected to have familiarity with either C or C++ before entering this course. Note that no C++ code will be used, we will only be discussing C, so knowledge of C++'s more advanced structures (such as classes, templates, etc.) is *not* strictly necessary.

Textbooks and Other Resources

- The Racket Guide (https://docs.racket-lang.org/guide/index.html)
- Beautiful Racket (honor system payment: https://beautifulracket.com/)
- The Structure and Interpretation of Computer Programs (SICP) (available free online: https://mitpress.mit.edu/sicp)
- The instructor may also require readings from other online sources

Learning Groups

This course makes use of **learning groups** to facilitate education. You will be randomly assigned a new learning group every few weeks, and your instructor will let you know your new group via email. You are expected to:

1. Sit with your learning group during lecture.
2. Complete your part of learning group assignments, dividing problems amongst group members if requested by the assignment.
3. Share results from learning group assignment at the beginning of lecture with the rest of your group.
4. Work with your group on the in-class exercises during lecture.

If you have issues with any of your group members, you should let your instructor know so that groups can be reassigned.

Autograded Homework

Most programming assignments are automatically graded upon submission. The grading scripts are designed to mark all fully-correct programs as correct, and do the best job possible assigning partial credit where applicable. But under some cases, you may not receive as many points as you deserve. If you believe this to be the case, contact the course instructor so they can either fix the autograder to handle your case better, or manually grade your assignment.

You are only allowed 4 submissions on autograded homework. This is because your grade is received quickly after submitting, and it may be tempting to use the autograder as the only means of testing. Students are expected to test their own code before submitting. If you run out of submissions and need more, then contact the instructor.

> **Warning**
> Autograded assigments are reviewed for plagarism a few weeks after the deadline, not when graded. While copying an assigment from another student may give you a good grade in the short term, the long term consequences could be severe. Plagarism is taken very seriously on these assignments.

Grading Policy

This course is worth 1000 points. The points are allocated as follows:

1. **Two Exams**. 170 points each. Total 340 points.
   - Exam 1 will cover Programming Language Concepts and Python
   - Exam 2 will cover Lambda Calculus, Regular Expressions, and Parsing
2. **Two Projects**. Total 350 points.
   - Language Explore Project (100 points)
   - SlytherLisp Project (250 points)
3. **Three Homework Assignments**. Total 250 points.
   - HW 1 — Lambda Calculus (70 points)
   - HW 2 — Functional Programming (70 points)
   - HW 3 — Lambda Calculus Beta Reducer (110 points)
4. **Learning Group Participation**. 60 points.

Your grade letter will be derived using the standard plus-minus grading scale. In other words, if you have accumulated X points during the course of the semester, then your grade letter will be:

Grade	**If X is...**
A+[3]	$970 \leq X$
A	$930 \leq X < 970$
A-	$900 \leq X < 930$
B+	$870 \leq X < 900$
B	$830 \leq X < 870$
B-	$800 \leq X < 830$
C+	$770 \leq X < 800$
C	$730 \leq X < 770$
C-	$700 \leq X < 730$
D+	$670 \leq X < 700$
D	$630 \leq X < 670$
D-	$600 \leq X < 630$
F	$X < 600$

The instructor reserves the right to move the scale down, but it will never move up.

Late Work Policy

This course uses a **slip day policy**. Every student starts the semester with **eight (8) slip days**.

For each 24 hours you turn in an assignment late, it will cost you 1 slip day to turn in an assignment. You will not get any points off, you just need enough slip days to turn it in. For example, if you have 3 slip days left, and you turn in an assignment 4 hours late, you will have 2 slip days left after turning in the assignment.

Slip days can only be used for homework assignments and for deliverables on projects. You cannot use slip days for quizzes or learning group activities.

Regardless of how many slip days you have, the following rules apply:

[3] For some reason, I cannot put this in Trailhead. But you will still get the bragging rights.

1. No more than 5 slip days may be used on a single assignment without the instructor's permission.
2. You must turn in all work by midnight on the Thursday of finals week, even if you still have slip days left.

Students are expected to keep track of their own slip day balance, and indicate the number of slip days they are spending on an assignment when they turn it in. For printed assignments, this should be at the top of the page, and for programming assignments, this should be in a comment at the top of the source or in a `README` file.

If you are running low on slip days and need more (for example, for a school-sponsored athletic event or medical condition) you are welcome to contact the instructor and ask for more.

> **Give me the TL;DR**
> You have eight slip days. Each 24-hours costs you one slip day. Keep track of your own slip days, and indicate the number you are spending when you turn in the assignment.

Collaboration Policy for Programming Projects in CS Courses

The following policy exists for all courses in the CS department. This policy is a minimum standard; your instructor may decide to augment this policy.

1. If the project is an individual effort project, you are not allowed to give code you have developed to another student or use code provided by another student. If the project is a group project, you are only allowed to share code with your group members.
2. You are encouraged to discuss programming projects with other students in the class, as long as the following rules are followed:
    a) You view another student's code only for the purpose of offering or receiving debugging assistance. Students can only give advice on what problems to look for; they cannot debug your code for you. **All changes to your code must be made by you.**
    b) Your discussion is subject to the **empty hands policy**, which means you leave the discussion without any record (electronic, mechanical, or otherwise) of the discussion.
3. Any material from any outside source such as books, projects, and in particular, from the Web, should be properly referenced and should only be used if specifically allowed for the assignment.
4. To prevent unintended sharing, any code stored in a hosted repository (e.g., on GitHub) must be private. For group projects, your team members may, of course, be collaborators.
5. If you are aware of students violating this policy, you are encouraged to inform the professor of the course. Violating this policy will be treated as an academic misconduct for all students involved. See the Student Handbook for details on academic dishonesty.

Disability Support

The Colorado School of Mines is committed to ensuring the full participation of all students in its programs, including students with disabilities. If you are registered with Disability Support Services (DSS) and your instructor has received your letter of accommodations, please contact your instructor at your earliest convenience so you can discuss your needs in this course. For questions or other inquiries regarding disabilities, we encourage you to visit Disability Support Services (DSS) for more information.

Calendar

This is a tentative schedule for the class. I have tried to avoid scheduling two assignments to be due on the same day. In cases where I was unable to do this, I scheduled a lab day on the day the assignments are due so you can have time to work on the assignments during class time.

Day	**Content**	**Available**	**Due**
January 8	Syllabus and Programming Language Concepts	LGA-01, LGA-02	LGA-01
10	*Scheduling Anomaly — NO CLASS*		
15	Language Evaluation and Typing Systems		LGA-02
17	Python Introduction	LGA-03	
22	More Python	LGA-04	LGA-03
24	OOP and Exceptions	LGA-05	LGA-04
29	Exam Review		LGA-05
31	**Exam 1**		
February 5	Slytherlisp Introduction	SL D1, LEP	
7	Lambda Calculus	HW 1	
12	*Career Day — NO CLASS*		
14	More Lambda Calculus		
19	*President's Day — NO CLASS*		
21	Memory Management	LGA-06	SL D1
26	Regular Expressions and FSA	LGA-07	LGA-06, HW 1
28	Parsing	SL D2, LGA-08	LGA-07
March 5	Racket Intro	HW 2	LGA-08
7	Finish Racket Intro	LGA-09	
12	Exam Review		LGA-09
14	Pattern Matching		
19	**Exam 2**	SL D3	
21	Lab Day	LGA-10	HW 2, SL D2
26	*Spring Break — NO CLASS*		
28	*Spring Break — NO CLASS*		
April 2	Lab Day	SL D4	SL D3
4	Macros		
9	Tail Call Optimization	SL D5, LGA-11	SL D4
11	*E-Days — NO CLASS*		
16	Language Explore Project Presentations		LEP Programs, LGA-11
18	Language Explore Project Presentations		
23	Language Explore Project Presentations	SL D6 & 7	SL D5
25	Lab Day		
30	Lab Day		
May 2	Lab Day		SL D6
7	*Finals Week — NO CLASS*		
9	*Finals Week — NO CLASS*		SL D7

Time Allocation

Please note that the first half of the semester (up to Exam 2) is fairly theoretical, while the latter half of the semester is more applied. Do not be blindsided by this transition.