SYLLABUS – CS 536: Foundations of Computer Science
Instructor: Rose Bohrer (pronouns: she/her)
Time: T,Th | 4:00 PM – 5:20 PM
Classroom: Higgins Labs 114
Instructor Office: Fuller Labs 139

Sections About People (The Important Stuff)

Inclusion + Classroom Climate Expectations

Computer science is for all of us. That means you too. It is important to take a moment to say this because I can guarantee there are students in this class who do not always feel welcome here. Maybe even you.

Inclusion only gets better when people work together to make it that way. I expect all of you to welcome each other, in your own way. On my part, I hope to create a classroom climate where you would feel comfortable sharing any feedback you have on this topic, and also feel comfortable reaching out for help if you do experience any ignorant behavior. Though I have not lived through all the things that my students live through, I know what it is like to look around the room and see that nobody looks like you. I will do my best.

To be clear, here is an incomplete list of things that need to be said more in CS classrooms: All Black Lives (Still) Matter, Anti-Asian Racism is a Pandemic, No Human is Illegal, and Disability Rights are Human Rights.

Mental Health (is in the Back)

I have put my mental health statement at the end of the syllabus to avoid triggering anyone. I encourage you to read it when you are in the right mood because mental health is more important than school.

Disability + Accommodations

The best way to arrange accommodations is to have OAS (Office of Accessibility Services) notify me – this helps me keep everything organized. It is standard to remind you of this on the syllabus.
OAS Email: AccessibilityServices@wpi.edu **OAS Phone:** (508) 831-4908

Do you want to know what accommodations are or whether you can get them? OAS and your doctor are good people to ask. I'm also happy to give my non-professional advice for things I personally live through: autism, ADHD, Ehlers-Danlos Syndrome, transphobia, and misogyny.

Is there an accommodation that would help you, but OAS does not usually provide? I would love to discuss with you. I can't promise anything, but we should try.

Stupid Question Policy

It is course policy that there are no stupid questions. If you still feel that your question is stupid, it is course policy that you should ask it anyway. This policy matters because when you need help, you may not be sure how to ask a perfect question yet. You may also have questions that relate to background knowledge or life outside of class, and these might be even more important. This is a list of questions that are not stupid. It is not complete (there are no stupid questions):

- What happened in the last two weeks of class?
- Is my project idea dumb?
- What do I do if English is hard?
- What does the school counseling center really do? Does it really even help?
- I'm having a hard time in school. What are accommodations? Do I count?

- How do I do well in school if I work two jobs and I'm still hungry?
- How am I supposed to be motivated when **<*waves hands wildly at the world in general*>**

I cannot do my job well if my students are afraid to ask for help.

Deciding Whether to Take the Course

What's This Course About?

Short answer: "Make your own little programming language." This is significantly different from past iterations, which focused on implementing a specific set of language features. This is a project-based, programming-based course, and the lectures are in service of your self-directed projects.

I reserve the right to change the lecture schedule to cover whatever topics you tell me to, as you discover what your project needs. The first part of the course focuses on core technical skills that will help you actually do the project: project planning, functional programming, parsing, data structures for representing program syntax, and how to evaluate your own project work. The second part of the course covers a broad range of perspectives on programming language. This does not mean that it covers debates like "Should we program in language X vs. Y?" Rather, I cover: what do different kinds of people care about when they think about programming languages? How do researchers in type theory, software engineering, social sciences, and humanities ask different kinds of questions about programming languages? If I'm feeling extra trans, I might do a lecture on how to apply queer and transfeminist theory to the design of programming languages. I am aware that a significant fraction of my students are also trans and/or queer, and you are welcome to be an enabler in this regard.

My professional training is in "logic in computer science," e.g., the mathy, theory side of programming language design. Though some lecture time will be spent on graduate-level theory material, it will be treated as enrichment material, and you can 100% succeed in this course if you're "not into CS theory".

What Will I Do In This Class?

You will do a group project about programming languages, typically a "design and implement your own language" project. Other than that: it's up to you. You will spend time picking your teammates, planning your work, and revising your plans as the project unfolds. I fully encourage students to pick and choose which aspects of the course they wish to dig into. For example, I would be ecstatic if one team focused on practical implementation, one focused on pure math, and one focused on pure critical theory.

Note on programming language choice: I will use a functional programming language (Scala) in my lectures. I will spend some lecture time on "How to Read Scala" – if you know several programming languages already, you are likely to pick it up just fine. If not, I encourage you to spend some extra time outside of class and turn to office hours early and often.
For your **own work**, you can use any programming language that you want: Python, Haskell, Jacquard loom, assembly code that runs on a toaster, whatever. I recommend functional programming languages because they make it fun and fast to implement other programming languages. However, this is seriously only a recommendation. If you're more comfortable in another language, just budget some time to deal with that language's unique challenges, then use it.

Why Should I Take It?

Everyone has different reasons, and that's totally fine by me:

- "Oh my god I just want to get into ONE (1) CS course this year, who cares which one"
- You will work as a programmer. Most people don't write compilers for a living, but most career programmers will use language design skills one day, because most big programs contain "little languages" – features that use these same design skills
- You work in this research area, or want to decide if that's something you'd like to do
- You're not into CS at all and the course gives you flexibility to do the work you enjoy
- You don't really think you'll use this stuff in real life but school's more fun if you just learn some random stuff because "vibes"

How Do Grades Work?

This course uses *ungrading*, meaning that you self-evaluate your learning and assess your own grade based on that. At the end of the course, you submit a form where you tell me what your final grade will be. Though I retain the right to override your grade decision, I have never reduced a grade – the point is that I reserve the right to bump up your grade if you're being unreasonably hard on yourself.

You will self-assess how well you meet the following list of course objectives, specifically:

- Identify a clear problem where programming language design can be used
- Communicate with clarity and technical depth about language design
- Develop a mathematically-precise definition of your language's syntax
- Work effectively with your team to plan your project
- Implement a well-scoped prototype in the language of your choice
- Situate your project among the schools of thought discussed in class

Is the Course Easy or Hard?

Before I answer this, I want to answer "is this course mean or kind?" My goal is to make the course as kind as possible. My mission is to support you through that material and help you grow, no matter where you're at. I use ungrading because grading would interfere with my mission of supporting every student. I hope you leave this course with a feeling that this course does not have to be scary or painful, and can be fun.

If I had to guess, the difficulty is comparable to when I teach CS 4536 and is less than my CS 3133 or CS 525. Students worry sometimes because I like lots of fancy topics that aren't discussed much in your previous courses. Compared to my other courses, I think this course makes it easier to lean on programming skills learned from previous courses, e.g., it's set up so you can succeed without doing theory stuff.

Course Details

Textbook(s) / Learning Materials

I use learning materials that don't cost you any money. This course is broad, so it will mix pieces from multiple textbooks, plus some research papers. If the long list looks overwhelming: I promise the stuff I actually want you to read will be manageable. The textbooks are free e-books and the papers are accessible free-of-charge through the WPI network.

- Scala: It's up to you to decide how much Scala you want to learn. If you will use it in your projects, you will want to learn it in more depth. If you just want to understand my lectures, you

can just learn the basics. The quickest, most basic resource would be the Scala cheatsheet:
https://docs.scala-lang.org/cheatsheets/index.html
Think of this as a quick resource that you can keep on hand for frequent use.

There are lots of resources listed at the address below. If you want to learn in depth, pick whichever resource you like:
https://docs.scala-lang.org/learn.html
If I had to pick one that covers what we need in class, I would say go to this one:
https://tourofscala.com/
And do the first 8 exercises, plus the "case class" and "companion objects" exercises.

- Regular expressions and context-free grammars: I teach these topics using lecture notes written by WPI's Prof. Dougherty, which I upload on the Canvas site.
- The **fastparse** library for writing parsers in Scala: My own notes, plus official documentation:
https://com-lihaoyi.github.io/fastparse/
- User studies
Homework will be based on Jonathan Aldrich's:
https://www.cs.cmu.edu/~aldrich/courses/17-396/assignments/asst7.pdf
https://www.cs.cmu.edu/~aldrich/courses/17-396/assignments/asst9.pdf
Lectures will be based on Michael Coblenz's notes:
https://www.cs.cmu.edu/~aldrich/courses/17-396/slides/HCPLD1.pdf
https://www.cs.cmu.edu/~aldrich/courses/17-396/slides/HCPLD2.pdf
https://www.cs.cmu.edu/~aldrich/courses/17-396/slides/HCPLD3.pdf
https://www.cs.cmu.edu/~aldrich/courses/17-396/slides/HCPLD4.pdf

- Critical studies
- For the remaining topics, the only reading is my lecture notes, which will be posted on Canvas as usual. E.g.: The precedence-climbing algorithm for parsing, types, and operational semantics

Do I Need Any Software?

You will need the programming language of your choice. I personally will use Scala in lecture.

Help Hours/Office Hours

For CS 536: Mondays, 4-5pm, in-person in FL 139

I also have several other help hours each week for my other students. They get priority at those times, but you are welcome to show up during those times if I'm free: Wednesday 3pm-4pm (in-person in FL 139) and Thursdays, 3pm-3:50pm (on Zoom: https://wpi.zoom.us/j/4019031992)

What are They? Office Hours, also called Student Hours, are time specifically set aside for the instructor (me) to give hands-on help to students (you). Office hours are one of the main and first places you should go if you would like help understanding anything in the course, both for help with a specific task or general help. One advantage of office hours is that you can ask for direct or detailed help with course work. Although we spend most of the office hours time talking about things from the course, they can also be a good opportunity for unstructured learning. If you are curious about anything such as related topics or current research in this area, feel free to ask.

What if I Need More Help? I know there will limited scheduled hours of office hours each week, so it can be hard to get enough individual help in a busy week. This will be especially true in the first half of the semester, when I am teaching CS 3133 and CS 536 at the same time. I want to help you despite these limits, so here is my advice for how to get help despite our limited resources:

- If my office door is open, you can always ask whether I am free to answer a question.
- You are always welcome to ask questions on the course Slack at any time. You are encouraged to ask questions early and often.
- You can post the question on Slack where other students can see it and even answer it. This also reduces the number of duplicate questions and helps me answer you faster
- I will check Slack at least once every weekday during business hours. I usually answer it much more, but questions asked at night are usually not answered until the next day.
- Using Slack helps me be more available to help you
- If you want more scheduled office hours, tell me. I cannot promise them, but I will try.
- If you want an individual meeting for 1-on-1 help, tell me. I may respond by scheduling general office hours instead, but it's good to ask because sometimes I can do 1-on-1.
- If you want a specific kind of office hours such as a review session, tell me.
- If you're not sure how to start, here are tips for how to ask questions that get good answers.

Schedule

Project Checkpoint Contents

The checkpoints are highly structured. They ask you to have regular team meetings and they include structured forms suggesting what to do before and during the meeting, and proposing a structure for the project work submitted each week. However, it bears repeating that “homework” in an ungraded course works very differently from “homework” in a traditional course.

At the end of the term, you will assess for yourself how well you met the learning objectives, which I use as your final course grade. The checkpoints are structured this way because, if you follow this structure, I believe you are likely to be happy with your self-evaluation, I believe you are likely to learn a lot.

If you and your team have discussed what works best for you and you choose to innovate on my checkpoint structure, or you choose to explore a different style of project from what I laid out – you should do that! You’re in charge here. Any structure I provide is a tool for you to use as you wish.
If you do pursue a radically different project type, it’s probably a good idea for someone on your team to interact with me regularly, in office hours or elsewhere, just so I can support you.

The last day of the term is a project celebration, where each team spends a couple of minutes introducing their project to the class, focusing on the positives of what their team did – these are casual introductions with no slides or demos, or specific preparation. Students will be provided a way to share compliments with each other about their projects. Given the small size of the course, this final day may end up being a short meeting, to give you time to work on your other courses.

Project Checkpoint Deadline Policy

Deadlines in ungraded courses don’t work the same way as graded courses. We don’t have extensions in the normal sense because (1) there’s no grade anyway, and (2) the goal of submissions is to get timely

feedback on the current state of your project. If your project isn't where you wanted it to be yet, just submit what you have so I can give feedback quickly.

If you do forget to submit on time, just send it to me once you remember, e.g., by email.

Attendance

Attendance is explicitly not required. You do not need to tell me when you have to miss class. This is an intentional decision because participation grades are bad for disabled students and others.

Your Final Grade

This course is ungraded. This means that you grade yourself based on how well your final project meets the learning objectives for the course. The final project handout on Canvas has the details.

I create an environment for you to learn; you tell me your grade at the end. I occasionally increase students' grades above their self-grade, if they're too hard on themselves. I have never reduced a grade below the one a student submits, and never intend to do so.

The only case where I would reduce your grade is if you consistently create a hostile or bigoted environment for other students after I warn you. This is what it means to value inclusion. I have never used this rule and expect it to stay that way

Where Do I Find The Course Content Online?

Canvas. I do not have a separate course website.

Technology Policy

Some days, I will work through examples on my computer in class, so you are welcome to follow along on your computer in class. A computer is not mandatory. You are welcome to goof off on your computer – I too was once guilty of doing homework for one class during a second class to stay awake, and it wasn't a bad class either. If you do use electronics for primarily entertainment purposes, try to sit somewhere that it won't distract your classmates, though.

Academic Integrity Policy

WPI's website has university-wide academic integrity policies. However, by the nature of this course, it is pretty much impossible to cheat. Pretty much the only way to cheat is if your team handed in another team's project, which I would notice very easily.

Mental Health

Content warning: transphobia, racism, xenophobia, suicide, and police brutality.

I started this job in Fall 2021. My first year here coincided with the peak of the worst mental health epidemic in WPIs history, with the deaths of 7 students in 6 months. Most of the deaths were by suicide and most of the victims were in computing-related majors.

On a personal level, I want my students to know this profoundly shapes how I view the job. When compared against your personal well-being, academic achievement ***simply does not matter to me at all.*** My favorite part of my job is the opportunity to support you as a person, and mental health is key to that. I don't want to just repeat some boilerplate, so here are some things I want you to know instead:

1. I have been there. There is nothing you can say or do that will scare me. I mean that in a good way.
2. Social connection is good for you, so I try to encourage social interaction in the course design.
3. I'm not going to judge you, period, I just want you to be okay. Missed class? Don't know what's going on? Cried in lecture? Vomited on my office floor? I'm not going to judge you. ***I need you to internalize this fact.***

I have been encouraged to tell you to contact the counseling center for non-emergency treatment (sdcc@wpi.edu, 508-831-5540, 16 Einhorn Road) or call Campus Police (508-831-5555) if someone's life is in immediate danger. However, nobody is stopping me from also telling you:

1. There are many therapists in Worcester that take student insurance
2. General practitioners ("your regular doctor") are the main source of antidepressant prescriptions.
3. Calling American police during mental health crises has gotten innocent patients killed, especially Black, Brown, trans, and disabled patients. They have traumatized people who I personally cherish. Mobile crisis response teams are an alternative and are often much better at avoiding violence. Community HealthLink has one in Worcester. I do not know whether it is good, but I feel I should share a link: http://www.communityhealthlink.org/chl/youth-and-family-services/youth-mobile-crisis-intervention-ymci
4. The Trans Lifeline (877-565-8860) is a crisis-intervention hotline ("suicide hotline") run by binary and nonbinary trans people. As of 2020, they do not call police unless required by law.

The family of one of the victims (Nathan S. Morin) called for donations to American Foundation for Suicide Prevention. In his memory, consider this a standing invitation to support mental health orgs in our community.

In conclusion: I think about mental health a lot. Like, a lot a lot. Don't be shy, okay?

What would useful industry stuff look like?
- Config files
- Code autogeneration
- Mini languages for specifying permissions / sets
- CSV

What would assignment timeline look like?

1. Project Planning
2. Syntax design + Parser-generation
3. ASTs + Pretty-printers + basic checks
4. Goals and evaluation and study results
5. What's it mean? What semantics should t be anyway?
6. What should its types be if any? Who cares?
7. Reflect
8. Queer it
9. Have fun

10. Done

Week 1: Have Fun, Pick Project Teams

- Introduction
- Topics + Teams

Week 2: Learn How to Plan and Evaluate PLs

- Evaluation of Languages
- Kinds of languages + impl approaches

Week 3: Syntax Review: RE + CFG

- RE
- CFG

Week 4-5: Learn (Functional?) Programming? In Scala?

- Fastparse
- Precedence climbing
- Datatypes
- HOFs
- Pretty-printing generators

Week 6: How to User Study – take from CMU class

Week 7: Study + Reflect + Plan

Week 8: Semantics

- Operational basics
- Operational hard version

Week 9: Types

- Boring type system
- Parametricity idk lol

Week 10: Dialectics

- Write my paper by now
- Explore what properties of your lang can/cannot be done
- At least a week of queer theory
- Feminist programming languages
- Why do trans women write rust
- Redo entirely

Week 11: Beauty

- Processing
- Penrose

- FARM papers
- Failure continuations as queer play

Week 12: Play

- Inform
- Twine

Week 13: Distributed

- Parallel semantics
- MLton Stefan stuff

Week 14: TBA+Present

My ideas:
1. Idea picking: Pop charades

2. Functional programming, why

3. Evaluation framework for languages

4. Rhetorical evaluation of languages

5. User studies for languages

Distributed?

This course discusses the fundamental concepts and general principles underlying current programming languages and models. Topics include control and data abstractions, language processing and binding, indeterminacy and delayed evaluation, and languages and models for parallel and distributed processing. A variety of computational paradigms are discussed: functional programming, logic programming, object-oriented programming and data flow programming. (Prerequisites: student is expected to know a recursive programming language and to have an undergraduate course in data structures.)

https://www.cs.cmu.edu/~aldrich/courses/17-396/

Jan 14	Course introduction
Jan 16	Concepts for Language Design

Jan 21	Project proposals
Jan 23,28	Formalism, e.g. Featherweight Java (paper), Evaluation and Typing Derivations (.txt) (view in fixed-width font)
Jan 30	Soundness; Assembly-like IR (see sections 2.2 and 3.1.3 of my program analysis book)
February 4	JavaScript; calculator example (calc.zip, web page)
February 6	Implementing an Interpreter in JavaScript (coverage of Assignment 6 constructs was deferred to Feb 11)
February 11	Big-step environment semantics
February 13	Introduction to User-Centered Programming Language Design
February 18	User-Centered Programming Language Design, part 2
February 20	User-Centered Programming Language Design, part 3
February 25	User-Centered Programming Language Design, part 4
February 27	Internal Domain-Specific Languages
March 3	Case Study: The Design of the Obsidian Language
March 5	Parsing
March 19	Pure OO PLs and Transpilation
March 24	Transpilation Demo (txt notes, manual c translation, generated c, transpiler impl)
March 26	Corpus Studies for Programming Languages
March 31	Case Studies on Programming Languages
April 2	Language Expressiveness
April 7	Truffle: Partially Evaluating Interpreters
April 9	Glacier: Usable Enforcement of Transitive Immutability

April 14	Penrose: From Mathematical Notation to Beautiful Diagrams
April 16	Discussion: User Study results
April 21	Wyvern: Designing a Language for Security
April 23	The History and Impact of OO: Simula and Smalltalk
April 28	Programming Language Design and Performance. See also this lecture in my program analysis course for an example of getting high performance from using a DSL.
April 30	Perspectives on PL Design and Course Conclusion.
May 7	Final project presentations

Fisler (20 YEARS AGO):

2x: overview

2x: first-order, data rep

2x: impl first-order

2x: closures, mutation, recursion

2x: recursion, exceptions, contexts, letcc

2x continuations

2x: semantics, lambda calculus

2x compilation

4x garbarge collection

2x types

2x type inference

2x polymorphism

2x scripting

https://cseweb.ucsd.edu/~mcoblenz/teaching/291I_fall2022/