little languages

Welcome!

- **Over half of you have already submitted Problem Set 0 and we haven't even given a real slide yet!**
- Trying to get in? This pilot course is currently oversubscribed and I do not anticipate being able to take anyone else on, unfortunately.
- I am scheduled to offer this course again in the Fall with the enrollment doubled.

Who am I?

- Kris Jordan
- Teaching Professor since 2015
  - Graduated from UNC with a BS in Computer Science in 2007
- This course was a missing course for me when I graduated…
  - … and it's been a missing course in the department since.
- I'm really excited to bring this course to life and believe you're the best possible group to help me pilot his course.

Why should *you* take this course?

- Computer scientists and software engineers depend on **tools** and **utilities** frequently in their day-to-day work:
  - Command-line shells
  - Project build systems
  - Regular expressions
  - Modal text editors
  - Documentation markup languages
  - Version control systems

- Wielding these tools well is a force multiplier when solving real software engineering, task automation, and data processing pipelines.

Each of these tools has its own "Little Language"

- The phrase "Little Language" is a term coined by Jon Bentley (UNC '76):

  "Languages surround programmers, yet many programmers don't exploit linguistic insights. Examining programs under a linguistic light can give you a better understanding of the tools you now use, and can teach you design principles for building elegant interfaces to your future programs."

This course will not teach you **all** of the tools...

- Or even *most* of them. Knowing them all, like knowing every programming language or every spoken language, is impossible.
- This course will teach you how to think about little languages, their structure, and how to implement your own.
- We will structure the course around case studies of the enduring, brilliant, and historically significant little languages most computer scientists and software engineers make use of **frequently** today...

  ...most of which were invented in the **70s**!

  - Thus, you *will* leave with a solid handle on important tools that can improve your productivity and enable you to automate tedious tasks away.

Discussion: Form groups of 2 or 3

1. Introduce Yourselves!
    - Name, year, what other courses you're taking.

2. Write down up to 4 of each of the following two categories:
    - Little Languages BOTH or ALL THREE of you have written and had a computer interpret.
    - Little Languages ONLY ONE of you has written and had a computer interpret.

3. Have 4 of each? Rank order the ones everyone uses in terms of value/frequency/importance.

- You'll have 4 minutes and then we'll come back together as a group.

Little Languages Discussion

pollev.com/compunc

Survey on Academic Dishonesty

- The department is trying to get a better sense of how students experience and perceive academic dishonesty among peers, if at all.

- Please respond to the following survey and check-in on PollEv when completed:

http://bit.ly/cs-cheating-survey

- Check-in on **PollEv.com/compunc** when complete

Meet your Team

- Helen Qin
- Tabatha Seawell
- Duncan Britton
- Hank Hester
- Jay Randolph
- Brooks Townsend

Graded Components of the Course

- Quizzes and Participation - 10%

- Problem Sets, Guided Reading Questions, and Worksheets - 30%

- 2x Midterms - 20% per - tentatively:
  - Monday, February 18th
  - Monday, April 1st

- Final - 20% - Thursday, May 2nd, at 12pm

Collaboration Policy

- General Course Content – Collaborate away!

- Problem Sets, GRQs, Worksheets, no:
  1. Sharing code or letting a peer or someone outside the staff view your code
  2. Use shared code or view someone else's code
  3. Type on a peer's keyboard or let anyone else type on yours

- With proper citation in the headers (add a collaborators line), you are allowed to:
  1. Discuss high-level concepts, approaches, and pseudo-code ideas on whiteboards
  2. Help debug a peer's code by viewing their screen under the following conditions:
     1. Your own laptop must be fully closed and you may not share any code or "tell them what to do"
     2. You may not touch their keyboard
     3. They should do 80% of the talking, your 20% should be asking questions

Course Website

https://comp590-19s.github.io/

All grading will be done via Gradescope.

How to get help?

- Piazza
  - Link in footer of course site.
  - Think of it as a class wide study group, don't expect immediate answers.
  - Please try Googling your question first!
  - If you're answering, answer no differently than if you were speaking face-to-face.

- Office Hours via Course Care
  - Link in the footer of the website
  - We will post the help schedule to course care soon
  - If you got stuck in your initial install, we will be going back to Sitterson 008 today and Friday to help.

Course Virtual Machine

- Most of you have the virtual machine installed and ready to go. The rest of you should tonight or tomorrow!
  - Soft deadline: Friday
  - Hard deadline: Monday

- We'll work in a Linux environment all semester and with a variety of tools that are preinstalled on the virtual machine.
  - You'll learn to work with the vim text editor!

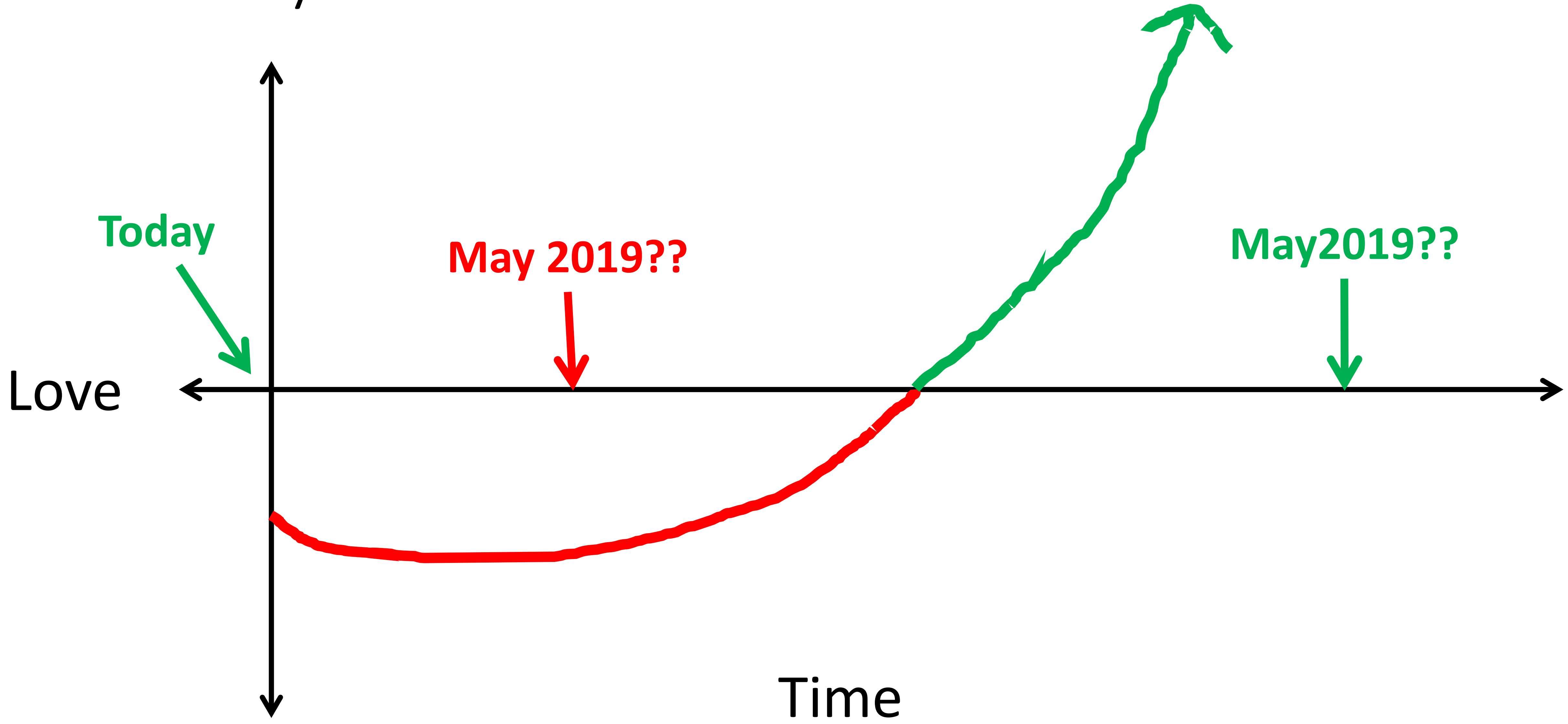
- The experience you gain working in this environment will pay dividends in the rest of your career at UNC and beyond.
  - You'll be comfortable working at the command-line
  - You'll learn to work with the vim text editor!

How you feel right now.

How you'll feel for the first monthish…

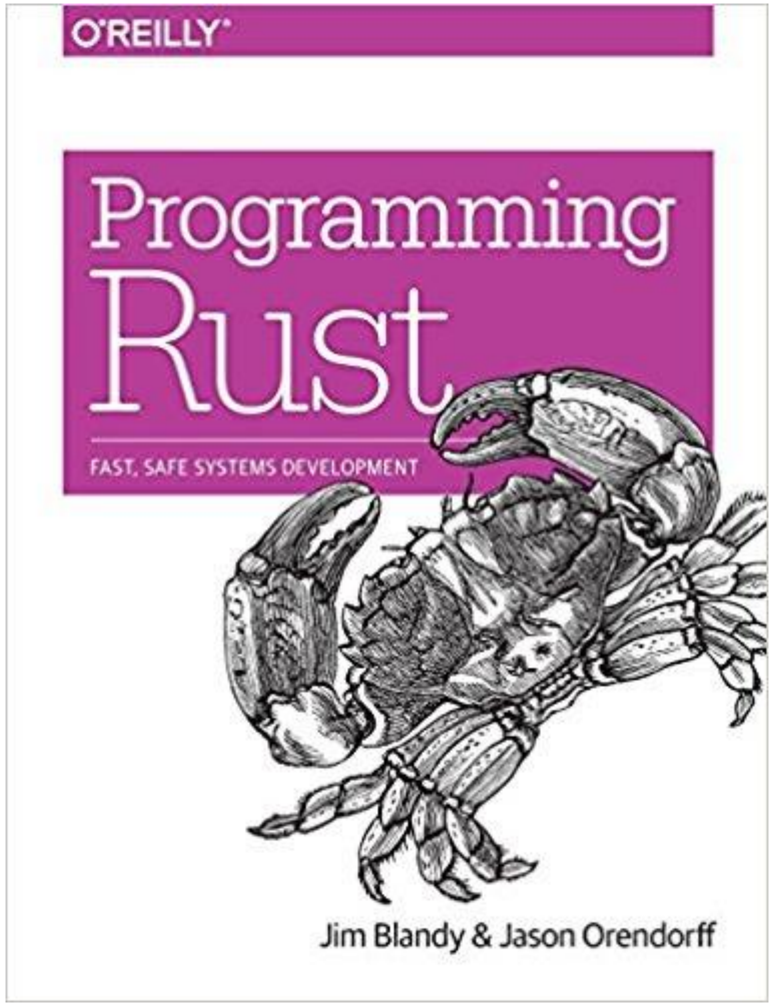
How you'll feel in May.

Your love, enjoyment, and appreciation of the work you'll do this semester...



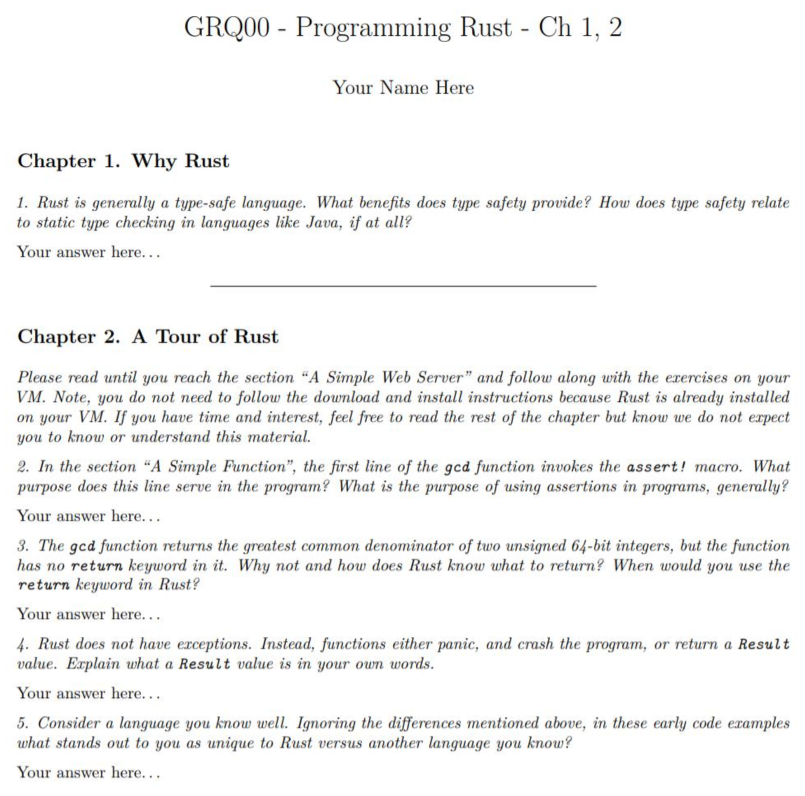
Course Text

- Readings will be regularly assigned, starting today.
- These will be from the official course book "Programming Rust" by Blandy and Orendorff
- Be sure your edition is at least 1st Edition Third Release (2018-06-22)



Guided Reading Questions (GRQs)

- When readings are assigned there will be Guided Reading Questions provided, as well.
- The instructions for working on GRQs is on Resources > Course Materials & GRQs
- You will edit these in a little language called Markdown on the Virtual Machine.



GRQ00 - Programming Rust - Ch 1, 2

Your Name Here

Chapter 1. Why Rust

1. Rust is generally a type-safe language. What benefits does type safety provide? How does type safety relate to static type checking in languages like Java, if at all?

Your answer here...

Chapter 2. A Tour of Rust

Please read until you reach the section "A Simple Web Server" and follow along with the exercises on your VM. Note, you do not need to follow the download and install instructions because Rust is already installed on your VM. If you have time and interest, feel free to read the rest of the chapter but know we do not expect you to know or understand this material.

2. In the section "A Simple Function", the first line of the `gcd` *function invokes the* `assert!` *macro. What purpose does this line serve in the program? What is the purpose of using assertions in programs, generally?*

Your answer here...

3. The `gcd` *function returns the greatest common denominator of two unsigned 64-bit integers, but the function has no* `return` *keyword in it. Why not and how does Rust know what to return? When would you use the* `return` *keyword in Rust?*

Your answer here...

4. Rust does not have exceptions. Instead, functions either panic, and crash the program, or return a `Result` *value. Explain what a* `Result` *value is in your own words.*

Your answer here...

5. Consider a language you know well. Ignoring the differences mentioned above, in these early code examples what stands out to you as unique to Rust versus another language you know?

Your answer here...

First Assignment Deadlines

- PS00 – Hello, World
    - Sunday 1/13 at 11:59pm

- GRQ00 – Ch 1 and Ch 2 (thru Handling CLI Arguments)
    - Monday 1/14 at 11am

We’d love **feedback** throughout the semester.

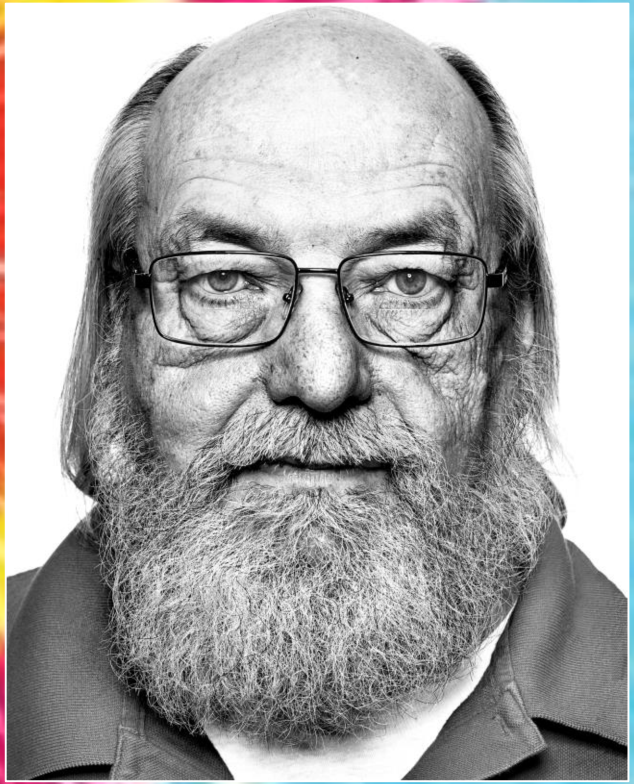
- I welcome feedback on all aspects of the course
- Feel free to email me feedback directly or share with a UTA.
- **Please give us feedback while we have time to act on it!**
- I’ll also take class wide feedback through the semester.

How does this course relate to others?

- Every Course
  - Every course has tools at work behind the scenes. Some hide them more than others. The more you understand about how they work they more effective you'll be at solving problems.
- COMP431 – Networking
  - Protocols are little languages that need to be parsed
- COMP455 – Theory of Automata
  - We will make concrete and real the foundational ideas of 455
  - My dream: many of you saying "Ohhhh… so that's why a non-deterministic finite state machine is useful." or "Wait. Regular expressions are useful?"
- COMP520 – Compilers
  - Building a true programming language compiler requires techniques you'll see first here and gain far more depth on in 520. You'll also handle semantic analysis (type checking) and generate machine code in a way we will not.
- COMP530 – Operating Systems
  - You will gain foundational concepts of OS APIs in this course which will help you succeed in 530.

Protagonists in our Hero's Journey through the Psychedelic 70s of Unix's Little Languages

Lorinda Cherry
aka Pipeline Queen



Ken Thompson
aka Gandalf of Unix