

While you wait

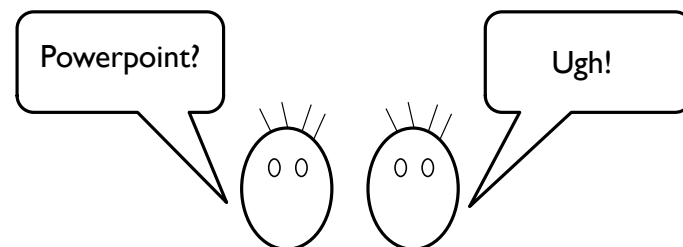
- <https://cs110.students.cs.ubc.ca/lectures/110-intro.pdf> <<< Slides!
- <https://cs110.students.cs.ubc.ca/admin/links.html>
- Follow Setup link
 - skip to installing DrRacket
 - but stop when you get to “setup test file”
 - that will let you type at DrRacket for class
 - go back after class and do skipped setup page steps IN ORDER
- Or, do all this after class and just use pen and paper during class!



CPSC 110

Systematic Program Design

- Who, Why, What and How
- Start working on the first module
- Don't worry – this is not a powerpoint course



Who?



Gregor Kiczales (he/him/his)
Professor of Computer Science
Fellow, Association of Computing Machinery

Research in programming languages and program design

Common Lisp Object System (CLOS)

Portable CommonLoops (PCL)

Metaobject Protocols (MOPs)

Art of the Metaobject Protocol

Aspect-Oriented Programming (AOP)

AspectJ

EdX Systematic Program Design (aka How to Code) courses

Who?



Dr. Oluwakemi Ola (she/her)

Assistant Professor of Teaching

Instructor for Vantage section

Research Focus: Information Visualization, CS Pedagogy

Who?



Emily Fuchs
Course Coordinator

cpssc110-admin@cs.ubc.ca

<<< administrative question go here

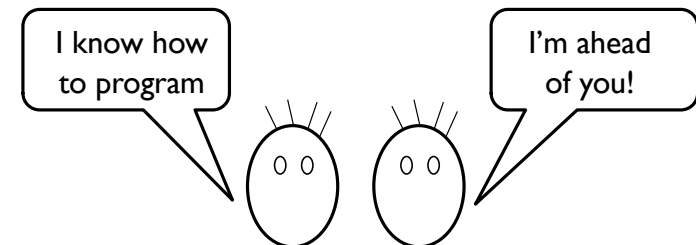
Technical questions go to Piazza

only sensitive personal issues go to instructor email

+ 50 TAs who work on labs, office
hours, Piazza, and grading

Who? (you)

- By year:
 - 1st 58%
 - 2nd 22%
 - 3rd 14%
 - 4th 3%
- By program
 - BSC 55%
 - BA 29%
 - BCOM 6%
- No programming experience required!
 - people with programming experience usually catch up to people with no programming experience in 3-4 weeks



What?

- Foundations of software engineering
 - more than just programming



Margaret Hamilton accepting US Presidential Medal of Freedom

Key ideas (1/5)

quickly today
but you will hear these
throughout the course

- Code that works properly is not enough
 - Need to be able to explain why you are confident it works properly
 - Need to be able to reliably produce similar programs
- Software is a team activity
- How we work determines what we produce
- Representing information as data
- Programs have structure

Key ideas (2/5)

- Code that works properly is not enough
 - Software is a team activity
 - useful programs always have to be modified by other programmers later
 - being kind to other developers is essential to success
 - applies to the code we produce, the questions we ask, the answers we give
 - How we work determines what we produce
 - Representing information as data
 - Programs have structure
- kindness starts on Piazza!

Key ideas (3/5)

- Code that works properly is not enough
- Software is a team activity
- How we work determines what we produce
 - we can't rely on jolts of brilliance
 - ACM Code of Ethics 2.1
Strive to achieve high quality in both the processes and products of professional work.
- Representing information as data
- Programs have structure

Key ideas (4/5)

- Code that works properly is not good enough
- Software is a team activity
- How we work determines what we produce
- Representing information as data
 - information is out there in the world
 - programs operate on data inside the computer that represents that information
- Programs have structure

Key ideas (5/5)

- Code that works properly is not good enough
- Software is a team activity
- How we work determines what we produce
- Representing information as data
- Programs have structure
 - of different kinds
 - local and crosscutting
 - being able to work in terms of that structure is powerful

Systematic Program Design (SPD)

- Systematic program design is a way of working that reliably produces well written, consistent, and well tested programs.
- Based on research and practice in programming languages and software engineering.
- Provides a foundation for professional software development
- Also relevant if you are NOT intending to be a software developer
 - helpful for programs of all sizes, including 2 page quick programs
 - underlying ideas help with all kinds of problem solving and design

Beginning Student Language (BSL)

- Programs are written in different languages
 - There are 10s of thousands of languages; thousands in active use. Hundreds are popular.
 - No one language is the most useful, best etc.
- BSL is the core of essentially all other languages
 - allows us to focus on learning systematic program design
 - prepares you for learning other languages quickly
 - never say a university course taught a language
- Puts all students on level playing field

Learning by solving design problems

- In lecture/lab/problem-sets/homework you will be working through program design problems
 - The real goal is NOT to simply handin a working solution to the problems.
 - It is to learn to solve the problem on your own.
 - If we help you too much, if you look at the solution too soon, if you get help from a friend, then you won't learn how to solve them on your own.
 - You have to work through the difficulty on your own.
 - It will be difficult, you will get stuck. That's when the learning happens!
 - Watching your friend lifting weights doesn't make you stronger.

Course Components - Lecture

- Before lecture you will work through videos and problems on edge.edx.org
- Lecture will mix presentation of new material with you working on problems
 - “priming” enables situated learning of new topics
 - expect lecture to be difficult and tiring – experience doing real design
 - 15% of course grade is the work you hand in during and after lecture
- After lecture you will review material from lecture and work through additional videos and problems on edge.edx.org

Course Components – Lecture Starters

- Working in DrRacket on in-class lecture problems
 - work during lecture
 - submit several times for each problem
 - you will get autograder feedback
- Graded on progress according to recipe
 - you MUST make progress and submit starters during the lecture
 - you may continue working and you have until 10pm to improve
 - you will receive autograder feedback the whole way

Course Components - Labs

- Designing programs to solve more challenging problems
- Lab number n covers lecture module n ; so does problem set n
- Answering design review questions from TAs
 - about your lab work
 - about the prior week's problem set

Course Components - Problem Sets

- Close out each module with a problem set that assesses your mastery of all the material to date
- THE PROBLEM SETS PREPARE YOU FOR THE EXAMS
- Collaboration policy is in the Syllabus

READ IT! The consequences of academic misconduct are quite bad.

- Combined assessment:
 - automatic grading (autograder)
 - during lab a TA will ask you questions about how you designed the program

Course Components - Other

- Office hours - instructors and TAs (See Piazza)
- Midterms and final
 - assessment of your mastery of systematic program design
 - on campus
- Average of Midterm 2 and Final must be passing to pass the course
- There is no textbook, everything you need is on edge.edx.org
- <https://cs110.students.cs.ubc.ca/admin/links.html>

Grading Scheme

| Item | % of total course grade |
|------------------|--|
| Problem Sets | 15% |
| Labs | 10% |
| Lecture activity | 15% |
| edX questions | 0% - These are a good for your learning though, so do not skip them! |
| Midterm 1 | 15% |
| Midterm 2 | 20% |
| Final | 25% |

see <https://cs110.students.cs.ubc.ca/admin/syllabus.html> for critical additional points

110 vs. 103+107

- 110 is all of Systematic Program Design, in 1 term, using teaching languages
 - best and fastest foundation for being a major or taking CPSC 210 (Software Engineering in Java).
- 103 is about first 5-6 weeks of 110 in Python
- 107 is the last 7 weeks of 110
 - in the teaching languages, using 110 edX modules.
- **107 take the 110 final exam this term**
 - so equally hard but 103+107 is a more gradual pace

What it takes to do well

- Don't need math, STEM, etc.
- Do need:
 - attention to detail because a 1 character error can break a program
 - patience because it takes time to solve hard design problems
 - humility because simple looking problems can still be hard
- Many of you have attention to detail, patience, humility
 - athletes, musicians, gamers, artists, ...

“Course Contract”

- Course staff will provide
 - state of the art content based on research and practice in programming and software engineering
 - delivered using state of the art pedagogy in active and online learning
 - supported by significant investment in materials and resources
 - 50 person team, extensive office hours, rapid response to questions on Piazza
- You will:
 - work hard (12+ hours/week outside of scheduled lab & lecture times) and stay up to date – not get behind by even a day
 - trust the design recipes to get you to a solution
 - follow course rules of decorum and academic honesty

After Class

- <https://cs110.students.cs.ubc.ca/admin/links.html>
 - [do Setup](#)
 - [read Syllabus](#)
 - edX “after lecture 1”, and then “before lecture 2”