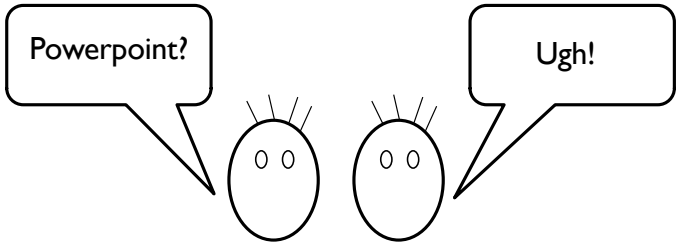
While you wait

- https://cs110.students.cs.ubc.cs/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?

Foundation for SPD is How to Design Programs (aka HtDP)

1st and 2nd editions

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi

Racket-lang.org

Racket, DrRacket, and many embedded tools

Above individuals plus many more

Who?

Systematic Program Design (our extension of HtDP)

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?

Jessie Pan
Course Coordinator

cpsc110-admin@cs.ubc.ca <<< administrative questions 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?

Emily Fuchs
Course Coordinator

cpsc110-admin@cs.ubc.ca

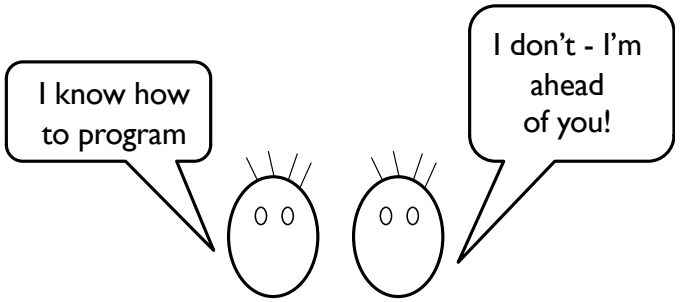
<<< administrative questions go here
Technical questions go to Piazza
only sensitive personal issues go to instructor email

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

Who? (you)

From a couple years ago

- By year:
  - 1st ~60%
  - 2nd ~20%
  - 3rd ~15%
  - 4th ~5%
- By program
  - BSC 50-60%
  - BA ~30%
  - BCOM ~5%
- No programming experience required
  - people with programming experience usually catch up to people with no programming experience in 3-4 weeks



What?

Margaret Hamilton accepting US Presidential Medal of Freedom

- Apollo moon mission software development lead
  - "How can I be sure the software will work so that people don't die?"
- Foundations of software engineering
  - more than just programming

Key ideas (1/5)

quickly today
but you will hear these
throughout the course

- Code that works properly is not nearly good enough
  - Need to be able to explain how you developed it
  - Need to be able to explain why you are confident it works properly
  - Need to be able to reliably produce similar programs
- Software development 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 nearly good enough
- Software development 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 nearly good enough
- Software development 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 nearly good enough
- Software development 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 nearly good enough
- Software development 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 design in terms of that structure is powerful

Systematic Program Design (SPD)

- Working systematically can reliably produce 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

What about ChatGPT (generative AI)?

What happens if you copy code from ChatGPT?

- Generative AI is going to end up playing a big role, but can you:
  - explain how you developed it?
  - explain why you are confident it works properly?
  - reliably produce similar programs?

- From someone who worked at Tesla: you would be fired
  - do you want (your) life critical code copied out of ChatGPT?
- In 110 it is academic misconduct – aka cheating

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 most other languages (lambda calculus)
  - 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 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.
  - It will be difficult, you will get stuck, your head will hurt – that's called learning.
  - Watching your friend lift weights doesn't make you stronger.

Academic Misconduct (Cheating)

- Cheating is stealing from other students and we won't tolerate it.
- Zoom poll right now!

A. I have already read and understood the syllabus and I know the rules of academic conduct in this course.

B. I will read the syllabus carefully tonight, learn the rules, and if I have any questions will ask on Piazza.

C. I will not check the syllabus, so I will risk breaking the rules – I know that not knowing the rules is not an excuse, so I could get into real trouble this way.

Course Components - Lecture

- Before lecture you will work through videos and problems on edge.edx.org
  - 10% of course grade is iClicker questions based on this material
- 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
- 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 submit to autograder to get feedback
    - based on whether you are working systematically
    - can submit as often as you like (within reason)
    - this is formative assessment (lecture starter grades don't count)

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!
  Again, cheating is stealing from other students and we won't tolerate it.
- Combined assessment:
  - automatic grading (autograder)
  - during lab a TA will ask you questions about how you designed the program

Course Components - Other

- Office hours - instructor and TAs (See Piazza)
- Midterms and final
  - assessment of your mastery of systematic program design
  - on campus
- Unweighted average of all three exams must be >= 50% to pass the course

  (MT1-grade + MT2-grade + final-grade)/3 must be >= 50%
- 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 questions (usually at start)	10%
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 one term
  - best and fastest foundation for being a major or taking CPSC 210 (Software Engineering in Java).
- 103 is based on first 4-5 weeks of 110, working in Python
  - 103 is a non-major course, less rigour, less depth
  - 107 is the last 7 weeks of 110
    - intended for 103 students who decide they want to major in CS
    - in the teaching languages, using 110 edX modules.
    - 107 students take the 110 final exam

What it takes to do well

- Don't need math, STEM, etc.
- Must have:
  - attention to detail because a one 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 (8 + 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
  - lectures page, lecture 01